Eighth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)

International Workshop on the Educational Uses of Multi-Agent Systems (EduMAS)

in conjunction with

International Workshop on Agent Based Systems for Human Learning and Entertainment (ABSHLE)

Budapest, Hungary

12th May 2009

Martin Beer, Maria Fasli and Debbie Richards (Editors)

http://www.windmill-cottage.net/EduMAS-09/index.html
Preface

There has been a sharp escalation in the number of applications and research questions related to the use of agent technologies to shape human experiences in complex environments. Increasingly we are finding that autonomous agents and multi-agent systems (MAS) have relevance to other fields both within and outside the Computer Science discipline. Consequently, there is a large number of people who want to learn about multi-agent systems, ranging from students, researchers in various fields, professionals working in industry or organizations, through to people that just want to understand more about, for instance, how agents bid in Internet auctions, how agents recommend products or how agents cooperate to arrange meetings for their human masters.

With the need for education and training in the field increasing, over the last few years we have seen both researchers and educators developing introductory and follow-up courses on multi-agent systems, platforms and tools to be used in such courses as well as text books, including a number that are now widely used. Also the increased sophistication of the multi-agent systems software now becoming available is allowing much more complex and interactive practical exercises and coursework to be attempted. However, all these efforts have generally been isolated and face a number of problems, namely:

- the great diversity of topics within the field of multi-agent systems
- the consequently heavy influence of one's own research expertise and specialization in deciding on the curriculum for a course
- the lack of venues for persons interested in the teaching aspect of the discipline multi-agent systems to exchange ideas and to learn from each other
- hostility from other more established fields in the Computer Science discipline, particularly with the recent large drop in student numbers and the move to a much coarser grained course structure

We are also concerned with the use of agents as a means to assist learning. A number of tensions accompany the use of agents in these contexts, since the goal is not to simulate autonomous agents for their own sake, but to use them to create an interactive experience with a pre-defined goal for the human user: either to learn a curriculum or to experience an engaging and rich world (or both, in the case of "edutainment"). Unlike fully author-controlled experiences such as such as films and plays, or fully scripted computer-aided instructional systems, dynamic interactive experiences require a world that can appropriately and meaningfully respond to the user - a natural fit for intelligent and believable agents. At the same time, however, system designers want to shape users' experiences, presenting new research challenges to address the interplay between player autonomy and designer intent. Thus, within this area of research, there is a design space
that ranges from complete autonomy for agents to complete control for an agent coordinator. One of the goals of this workshop is to foster a dialogue among researchers who are exploring the complex trade-offs that must be made in designing agent systems for education and interactive entertainment, and especially to bring together researchers focusing on autonomous multi-agent systems with those focusing on more centralized agent coordination for this problem.

The International Workshop on the Educational Uses of Multi-Agent Systems (EduMAS) provides an excellent forum to discuss these topics and we are very pleased to join with the workshop on Agent Based Systems for Human Learning and Entertainment (ABSHLE) to create an exciting program that will bring participants together and provide the means for stimulating discussions and exchange of ideas. We also welcome Forrest Stonedahl from Uri Wilensky's NetLogo team at Northwestern University for a plenary session describing his experiences both developing and using this major educational resource.
Workshop Organization

Programme Chairs

Martin Beer, Communications & Computing Research Centre, Sheffield Hallam University, UK.  email: m.beer@shu.ac.uk
Maria Fasli, UK. School of Computer Science & Electronic Engineering, University of Essex, email: mfasli@essex.ac.uk
Debbie Richards, Macquarie University, Australia. email: richards@ics.mq.edu.au

Programme Committee

Luis Antunes, Universidade de Lisboa, Portugal
Juan Carlos Augusto, University of Ulster, UK
Joerg Denzinger, University of Calgary, Canada
Rachel Goshorn, Naval Postgraduate School, USA
Michael Jacobson, University of Sydney, Australia
Petros Kefalas, CITY College, Greece.
James Lester, North Carolina State University, USA
Susan Lowes, Teachers College, Columbia University, USA
Simon Lynch, University of Teeside, UK
Eleni Mangina, University College Dublin, Ireland
Riichiro Mizoguchi, Osaka University, Japan
Antonio Moreno, URV, Spain
Ana Paiva, Instituto Superior Tecnico, Portugal
Agostino Poggi, University of Parma, UK
Colin Price, University of Worcester, UK.
Ilias Sakellariou, University of Macedonia, Greece
Elizabeth Sklar, Brooklyn College of the City University of New York, USA
Demosthenes Stamatis, TEI, Thessaloniki, Greece
Ioanna Stamatopoulou, CITY College, Greece & University of Sheffield, UK
Rainer Unland, University of Duisburg-Essen, ICB, Germany.
Harko Verhagen, Stockholm University, Sweden
Table of Contents

Plenary Presentation

NetLogo: Meditations on a Tool for Learning and Modeling
Forrest Stonedahl 3

Full Papers

Bridging the Gap: Introducing Agents and Multiagent Systems to Undergraduate Students
Emma Bowring, Milind Tambe 7

The SEREBRO Project: Fostering Creativity through Collaboration and Rewards
Rose Gamble, Sandip Sen, Bradley Brummel, Frank Grove, Noah Jorgenson, David Baker, Dan Gurernsey, Daniel Hampton, Jordan Hughes 15

Enhancing Training by Using Agents with a Theory of Mind
Maaike Harbers, Karel van den Bosch, John-Jules Meyer 23

A Framework for Teaching Multiagent Systems
Keerthi Rajendran, Simon Lynch 31

Allowing Trainers to Author their own Learning Content
Debbie Richards, Meredith Taylor 39

Multi-Agent System Coursework Design in NetLogo
Ilias Sakellariou, Petros Kefalas, Ioanna Stamatopoulou 47

Teaching AI and IA to non-Science Graduates
Ioanna Stamatopoulou, Maria Fasli, Petros Kefalas 55

Re-conceiving Introductory Computer Science Curricula through Agent-Based Modeling
Forrest Stonedahl, Michelle Wilkerson-Jerde, Uri Wilensky 63

Short Papers

Crafting a Personalized Agent-Oriented Mobile E-Learning
Olapeju Ayoola, Eleni Mangina 73

Functional Behavior of Traffic Control Systems Learned Through Multiagent Systems
Aleš Janota, Juraj Spalek, Jana Šebeňová 77
Plenary Presentation
NetLogo: Meditations on a Tool for Learning and Modeling

[ plenary talk abstract ]

Forrest Stonedahl
Center for Connected Learning and Computer-Based Modeling (CCL)
Northwestern University
Evanston, Illinois, U.S.A.
forrest@northwestern.edu

Keywords
NetLogo, Agent-Based Modeling, Multi-Agent Systems, Education, Modeling Environments, Programming Languages

1. INTRODUCTION
NetLogo [2, 3] is a premier agent-based modeling language and development environment, distinguished by such features as a gentle learning curve, friendly user interface, well-established user-base, and large library of classic and exemplary models. It is freely available for download from http://ccl.northwestern.edu/netlogo/. One of NetLogo's significant achievements is that it has come to play a central role in teaching numerous students and researchers from many fields about multi-agent systems and complexity science.

2. MEDITATIONS ON THE PAST
NetLogo comes from a rich tradition of learning-oriented software, being a descendant of the Logo programming language[1], which was designed to teach children about computer programming and mathematics. As a result, NetLogo shares Logo's twin design goals of having both a “low threshold” for entry, and a “high ceiling” in terms of expressive power and capabilities. In this talk, I will briefly outline NetLogo's heritage and how it has become what it is today.

3. MEDITATIONS ON THE PRESENT
The NetLogo software has been downloaded over 170,000 times, and is used in schools and research labs across the globe. Since its inception it has grown faster, more powerful, and easier to use. I will review the current state of the modeling platform, discussing or demonstrating features including network-based modeling, 3D modeling, participatory simulations and the HubNet architecture, and the extensive collection of educational models that are included with NetLogo.

4. MEDITATIONS ON THE FUTURE
The study of multi-agent systems and agent-based modeling is growing, and as it grows it evolves. As a result, the tools and techniques for designing, understanding, and analyzing multi-agent systems are co-evolving to meet the needs of the research community. In this talk I will discuss some of the current/on-going work to improve and extend the NetLogo language and environment.

5. CONCLUSION
This presentation will focus on NetLogo but is also intended to stimulate a broader discussion about the current state of tools and resources for agent-based modeling, and for education about emergence and multi-agent systems. Such a discussion should prove beneficial to all practitioners in this field, and help us to improve the software that is available for learning about complex systems and agent interactions.

6. ACKNOWLEDGMENTS
I am privileged to present these topics on behalf of NetLogo's author, Dr. Uri Wilensky, as well as the NetLogo development team, and everyone at the CCL research lab who has contributed to the ideas, models, and software discussed in this talk.

7. REFERENCES
Full Papers
Bridging the Gap: Introducing Agents and Multiagent Systems to Undergraduate Students

Emma Bowring
University of the Pacific
Stockton
CA 95211, USA
ebowring@pacific.edu

Milind Tambe
University of Southern California
3737 Watt Way, PHE 410
Los Angeles, CA 90275, USA		
tambe@usc.edu

ABSTRACT

The field of “intelligent agents and multiagent systems” is maturing; no longer is it a special topic to be introduced to graduate students after years of training in computer science and many introductory courses in Artificial Intelligence. Instead, time is ripe to face the challenge of introducing agents and multiagents directly to undergraduate students, whether majoring in computer science or not. This paper focuses on exactly this challenge, drawing on the co-authors’ experience of teaching several such undergraduate courses on agents and multiagents, over the last three years at two different universities. The paper outlines three key issues that must be addressed. The first issue is facilitating students’ intuitive understanding of fundamental concepts of multiagent systems; we illustrate uses of science fiction materials and classroom games to not only provide students with the necessary intuitive understanding but with the excitement and motivation for studying multiagent systems. The second is in selecting the right material — either science-fiction material or games — for providing students the necessary motivation and intuition; we outline several criteria that have been useful in selecting such material. The third issue is in educating students about the fundamental philosophical, ethical and social issues surrounding agents and multiagent systems: we outline course materials and classroom activities that allow students to obtain this “big picture” futuristic vision of our science. We conclude with feedback received, lessons learned and impact on both the computer science students and non-computer-science students.

1. INTRODUCTION

Since the first international conference on multiagent systems, ICMAS, held in 1995, to International conference on agents and multiagent systems (AAMAS), 2009, the entire field of “intelligent agents and multiagent systems” as represented by AAMAS has matured significantly. In earlier years, students were introduced to this field as a special topic, only as a graduate course, after years of training in computer science, and after introductory courses in Artificial Intelligence. Even the foundational principles of our field were unclear, and thus the only available syllabus was a set of ad-hoc line course materials and classroom activities that allow students to obtain this “big picture” futuristic vision of our science. We conclude with feedback received, lessons learned and impact on both the computer science students and non-computer-science students.

Over the years, our field has gradually matured, and we are in a historic transition period. This is similar to fields such as robotics and software engineering that have matured to a point where under-graduate students in computer science are able to take courses to develop relevant skill-sets in these fields. Agents and multiagent systems have similarly reached that critical mass — it is now time to face the challenge of teaching intelligent agents and multiagent systems directly to undergraduate students. In fact, given the potential social impact of our field — society in general will need to interact with agents and multiagents on an increasing basis — introducing the fundamentals of our field to non-computer science (and non-engineering students in general) is also important.

This paper focuses on exactly this challenge, outlining three key issues that must be addressed. The first issue is facilitating students’ intuitive understanding of fundamental concepts of multiagent systems. There are two obstacles: (i) While there is an emerging consensus on what constitute such fundamental concepts[15], appropriate textbooks that match all our requirements are missing and (ii) we lack conceptual tools to appeal to students’ intuitive understanding. In this paper, we outline our approach to the lack of appropriate textbooks. More importantly, to address the issue of developing intuitive understanding of our science, we introduced science fiction materials and classroom games. Moreover, these tools — science fiction and games — allow us provide students with excitement about multiagents, instilling that sense of wonder.

The second issue is in selecting the right material, either science-fiction material or games: we outline several criteria that have been useful in selecting such material. In essence, we need to engage in an appropriate tradeoff conciseness of the material and its usefulness in the concepts taught. The key here is to ensure that a science fiction episode/story or game acts as “spice” to the main dish of the actual content from the field of agents and multiagent systems. Thus, these extra materials should not dominate our lectures, and should help rather than hinder teaching of the desired concepts. The third issue is in educating students about the philosophical and ethical issues that have surrounded agents and multiagent systems, as well as concerns about their future social impact. This is a crucial issue as it provides students with this bigger picture view of our field, educating them in the foundational debates such as the nature of intelligence. Science fiction is particularly important in allowing us to address this issue, providing a rich framework to construct exercises and discussions. For example, we outline an exercise based on a “trial of a robot” based on a science-fiction TV episode, that allows students to debate the nature of intelligence, future rights of robots, and potential liabilities that agents designers may face in the future. Providing students with more creative and well-rounded courses that touched on philosophy, ethical concerns, social concerns allowed us to attract a broader student audience that was keen on understanding the social context of computing.

The solutions outlined above have been practiced since 2006 via the coauthors’ teaching multiple courses at two different major uni-
versities in the United States to undergraduate students, both computer science majors and non-majors. For the computer science majors, we have taught three iterations of upper-division courses on multiagent systems. For non-majors, we have taught a variety of courses. This includes three iterations of “general education” courses, introducing concepts in multiagent systems, possibly inspiring students to take on computing and computer science. It also includes short seminar courses for “welcome week” where the short courses are intended to encourage faculty-student interaction, and “parents weekend”, where the goal to allow parents to immerse themselves in an undergraduate course by attending short typical lectures taught by faculty. In our experience, all our student audiences face similar challenges in understanding basic concepts in agents and multiagent systems; however, the appropriate details that are needed to be covered for these audiences differ. In the case of computer science majors, the key is to provide sufficient details so they could actually implement the concepts as computer programs; for other audiences, the emphasis, particularly given the number of lectures and their format, may be to provide an understanding of the core concepts. Indeed, the key techniques introduced below appear to be useful to help teaching both computer science majors and non-majors.

2. MULTIAGENT SYSTEMS FOR UNDERGRADUATE STUDENTS

In introducing multiagent systems to undergraduate students, there are two main obstacles: (i) lack of a choice of appropriate textbooks given our course requirements; (ii) lack of a common set of conceptual tools to provide the appropriate intuition and motivation for understanding key concepts. As far as a choice of appropriate textbooks, it is important to note that the existing set of textbooks on agents and multiagent systems provide a foundational contribution to our field[17, 13]. However, our courses provided a unique set of requirements. First, we needed to teach multiple audiences at different levels of preparation as we needed to introduce multiagent systems to undergraduates who were both computer science (CS) majors and non-majors and thus could not necessarily assume significant preparation in CS. Second, available textbooks did not necessarily cover key topics in multiagent systems that we wished to emphasize, such as teamwork, swarm behavior, distributed constraint optimization, behavioral game theory, coalition formation, agent-human interactions and others. Third, we wanted to include some cutting edge topics so students would recognize that not all questions in our field are settled. The goal was to allow students to understand that significant questions still remained unanswered and that potentially this provided an exciting opportunity for further study in our field.

Our unique set of requirements led us to the conclusion that we could not rely on any of the available textbooks; yet providing undergraduate students detailed mathematical research papers to read was simply inappropriate. To compensate for the textbook we wrote a detailed set of class notes in intelligent agents and multiagent systems. The key was to write these notes for students of potentially diverse backgrounds, and hence to start from the basics. For example, students were perhaps not familiar with basic decision theory; hence the notes started out introducing decision theory and risk averseness in decision theory — these concepts are difficult to discuss in the abstract. Our goal was to find a hook into something that the students could intuitively understand, thus making it much easier to discuss these concepts with the students. Two separate types of tools have helped in this regard. First, science fiction stories, TV episodes and movies, have provided a social and fictional context for the discussion of basic elements of intelligent agent design and for multiagent interactions. Many computer science students are already familiar with or fans of science fiction, and as such we build on this familiarity. For example, the notion of recursive agent modeling can be introduced in terms of science fiction episodes of a robot reasoning about a human’s view of that robot (see below), which students find much easier to relate to than an abstract introduction to recursive agent modeling.

Similarly, developing intuitions about the core in coalitional games or risk averseness is difficult in the abstract. Having students play games for something they value — we have found small or large chocolate bars to be an ideal incentive — allows these intuitions to be developed. For example, discussions of risk averseness can be initiated by first having some volunteer students decide whether they would prefer a chocolate for sure or gamble for two chocolates and a penny. Several students will prefer the sure single chocolate, even though the expected payoff of taking the gamble is higher. This can help in discussions of decision theory and risk averseness. We discuss specific types of games in the following and criteria for selecting them.

3. ISSUES IN USING SCIENCE FICTION AND GAMES

As discussed in previous section, we used science fiction material and chocolate games in order to introduce key multiagent concepts; we now explain key criteria used in selecting such material and games.

3.0.1 Science Fiction Material Selection

Our selection criteria for the science fiction material for introducing multiagent concepts included the following:

- It had to exhibit some topic of interest in the agents and multiagent systems arena, e.g. agent modeling, emotions, teamwork, agent interactions under uncertainty, etc. There had to be enough in-depth examination of this topic in terms of its importance, or the different agent capabilities it enables. In essence, the science fiction material had to enable students...
to build up some intuitive understanding of this topic, its relevance to agents and multiagent systems, and potentially some of the complexities that may arise in this topic. In other words, the material needed to help us “bridge the gap” in understanding key concepts in multiagent systems.

- The story or film clip had to feature the robot or AI as an active participant in the plot or story.
- The story needed to present the robot/agent in a positive light. There is already a lot of popular science-fiction presenting a negative view of robots — combating this perspective with a positive view of robots/agents was considered important in order to motivate our students. (Unfortunately, meeting this requirement for all of our topics of interest proved to be quite difficult, and in one or two cases, we had to relax this requirement.)
- The story had to be short enough (maximum 30 pages); if a movie clip was to be used, it had to be short enough to be shown in 5-10 minutes at the beginning of class.

We chose several short stories by Isaac Asimov from his collection “Robot Vision” [1]. In addition to the short stories, the book also includes several essays by Asimov that provided useful reading material for the big-picture futuristic vision of our field as described later. We also chose episodes from “Star Trek: The Next Generation” that focused on the issue of agents, robots and intelligence.

Course lectures highlighted key aspects of agent behavior and functioning. For example, in the Asimov story, “Little Lost Robot” a robot (Nestor-10) intentionally tries to hide among a group of similar robots, while a human tries to run tests to isolate it from other robots. Nestor-10 is just different from other robots in the rules it follows. In the story, Nestor-10 considers what the human believes the other robots will do in order to behave like all other robots — so it can blend in. Meanwhile, the human must devise tests that prevent Nestor-10 from blending in. In particular, she must infer the plan the robot is executing to pass the test; by observing the robot’s actions, she can infer the plan the robot is executing, which in turn reveals whether the robot is Nestor-10 or not. Unfortunately, initially the human fails to recognize Nestor-10 because it anticipates the human’s intentions, and foils those by continuing to successfully blend in. This story provides a fictional context for introducing basic concepts in agent modeling. In particular, the story provided three specific settings to investigate three particular aspects of agent modeling.

- The basic idea of a human trying to infer the plan a robot is executing by observing its actions provides an initial introduction to plan recognition.
- Nestor 10 must predict what other robots would do, and imitate their actions. This allows us to discuss how agents predict other agents’ behaviors.
- Nestor 10 must recursively model what the human believes other (non-NESTOR-10) robots would do, enabling a discussion about recursive agent modeling.

Similarly, the Asimov story, “RUNAROUND” is based on a robot facing conflicting directives, which in effect brings up the notion of intention reconsideration and conflicting commitments. We can explain the behavior in terms of Belief-desire-intention (BDI) concepts[18, 7], and understand how to avoid such conflicts.

<table>
<thead>
<tr>
<th>Science Fiction Story or Movie</th>
<th>Brief Synopsis</th>
<th>Intelligent Agents and Multiagent Systems Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runaround by Isaac Asimov</td>
<td>Robot is stuck running around in circles because of internal rule conflict</td>
<td>Agents based on “Beliefs, desires, intentions” (BDI)</td>
</tr>
<tr>
<td>The Enemy (episode from “Star Trek: The next generation”)</td>
<td>Humans and Romulans enter a dangerous game of who blinks first</td>
<td>Introduction to Game Theory</td>
</tr>
<tr>
<td>Descent Part I (episode from “Star Trek: The next generation”)</td>
<td>The robot “Commander Data” shows emotions</td>
<td>Agent Emotions</td>
</tr>
<tr>
<td>Little Lost Robot by Isaac Asimov</td>
<td>Robot must reason what human trying to find it thinks about it</td>
<td>Agent modeling or plan recognition</td>
</tr>
<tr>
<td>2001 Data by Vernor Vinge</td>
<td>HAL</td>
<td>Adjustable autonomy and safety</td>
</tr>
<tr>
<td>Minority Report</td>
<td>Clip showing robotics spiders</td>
<td>Multiagent teamwork</td>
</tr>
<tr>
<td>Fast times at Fairmont high by Bruce Sterling</td>
<td>Small sensors create a network</td>
<td>Distributed constraint reasoning</td>
</tr>
<tr>
<td>The Swarm by Bruce Sterling</td>
<td>Insect species</td>
<td>Multiagent Swarms</td>
</tr>
<tr>
<td>The Offspring (episode from “Star Trek: The next generation”)</td>
<td>Commander data creates an artificial offspring and must teach it.</td>
<td>Machine learning</td>
</tr>
<tr>
<td>Who watches the watchers (episode from “Star Trek: The Next generation”)</td>
<td>Human colonists must be transported via shuttles</td>
<td>Coalition formation</td>
</tr>
</tbody>
</table>

Figure 1: Science Fiction Material used and concepts introduced
We outline in Table 1 some of the science fiction materials used and the concepts that were introduced using them. In all of these instances, students were either asked to read the story before class, or shown a short clip from the episode or film during class. In this way, the story or film provided a context for a discussion about the key concept: i.e., why is agent modeling important, what are the difficulties in the problem; what are the key concepts offered in the filmic or textual “solution”.

One major advantage of using science fiction in teaching multi-agents was that it allowed us to introduce cutting edge topics, to instill that sense of wonder about our field and about computing. We therefore intentionally introduced topics such as agent emotions in our class.[12, 8]. To cover emotions, we have used different science fiction materials. For example, in one of our iterations, we used an episode showing Commander Data from the popular science fiction series “Star Trek: The next generation” getting angry, i.e. showing emotions. A short clip from the movie “I, Robot” was also useful in introducing emotions; in this movie a robot is seen to get angry, and a human wondering if it is useful for robots to have emotions (see Figure 2). The famous scene of the spiders from the movie “Minority Report” (see Figure 3) was used to introduce students to the concept of teamwork in general, and in particular, to the more cutting edge topic of distributed POMDPs.[10, 2].

To conclude, we established several criteria to select science fiction material for use in our classroom. We were often able to find science fiction material that met all our criteria; nonetheless, material that provided perfect fit for all our topics was not always available. In particular, the science fiction material shown in Figures 2 and 3 also illustrates some of the difficulties in meeting all our selection criteria. The short movie clip of angry robot in Figure 2 does not get into the usefulness of that emotion or why it might arise; its use in “bridging the gap” discussed above is thus somewhat limited. The spiders shown in Figure 3 do show teamwork and meet all our criteria except that they present a somewhat negative view of robots. However, on balance, this was sometimes the best material available to introduce the topic at hand.

3.0.2 Choosing Games

Our criteria for designing games to play in the classroom included the following:

- The game had to force students to reason about something of value, but not cost too much to the instructors.
- The game had to exhibit some key characteristic of the topic studied in the lecture, just as in our selection of the science fiction episode.
- The game had to be short enough, lasting just 5-10 minutes at the beginning or in the middle of the lecture.

As mentioned earlier, having students play games for chocolates turned out to be an ideal solution for having objects of value that were not a significant cost to the instructors. Some of our lectures thus started out with a game at the beginning, and then focused on lessons from the game to introduce initial concepts. For example, lectures on decision theory and risk averseness started out with a game involving a gamble for chocolates. As mentioned earlier, this game involved first having some volunteer students decide whether they would prefer a chocolate for sure or gamble for two chocolates and a penny. Playing out this game at the beginning of class led to a discussion how these choices were arrived at, and thus led into decision theory. Figure 4 illustrates a chart that is used to discuss students’ preferences and compare their attitudes to risk.

In some variations of the course offerings, we also used chocolate games for introducing game theory. For example, we started off introducing prisoner’s dilemma with the payoff function as shown in Figure 5. In particular, each pair of students was given a tex-
Prisoner’s Dilemma: With Candies

<table>
<thead>
<tr>
<th></th>
<th>Cooperate</th>
<th>Defect</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2 chocolates, 2 chocolates</td>
<td>No chocolates, 3 chocolates</td>
</tr>
<tr>
<td>B</td>
<td>3 chocolates, No chocolates</td>
<td>1 chocolate, 1 chocolate</td>
</tr>
</tbody>
</table>

Figure 5: Chocolate game to illustrate the prisoner’s dilemma game.

The students were then asked to choose to cooperate or defect. The students’ choices and their reasoning behind their choice then led into an introduction to game theory.

A key point to note is that in playing games, many different outcomes are possible. These diversity of outcomes often provide an important opening for deeper discussions, and introduction to more complex topics. For example, students may have different risk attitudes in the chocolate game used to introduce decision theory; this clearly opens up a discussion of differences in risk attitudes. Similarly, while the rational choice in the prisoner’s dilemma game is for both players to defect, in the classroom, students may not necessarily play this “rational strategy”. Indeed, in playing the prisoner’s dilemma game, students quite often cooperate rather than defect. Such cooperation not only allows us to underline the paradox of prisoner’s dilemma, but also provides an opening for further introduction of behavioral game theory.

In summary, we used chocolate games in introducing the following concepts:

- **Decision theory**: Introduce expected value and risk averse-ness as discussed earlier.
- **Game theory**: Playing prisoner’s dilemma using candies as discussed earlier.
- **Auctions**: Introduce Vickery auctions, sequential auctions using fake money and chocolates as items to be auctioned off. To ensure that the fake money had value, there was an exchange rate for the fake money for some small candies, so students would not always bet all their money on the chocolate auctioned.
- **Coalition formation**: To introduce concepts such as imputations and the core in coalitional games, we used chocolate games in class yet again; the students were given different characteristic functions in terms of chocolates and had to form coalitions. Multiple different groups of students were given different problems, e.g. in one case the core was empty, leading to very difficult negotiations.

One key principle used was to ensure that there were not too many different tools used in the same lecture. So for example, if a science fiction story or TV episode was used, then the key was to ensure that we did not simultaneously also use a chocolate game. Using too many such ideas could end up distracting students rather than helping them learn the concept of interest.

4. **THE BIG PICTURE: PHILOSOPHICAL, ETHICAL AND SOCIAL ISSUES**

The third issue that we need to address in teaching agents and multiagent systems to undergraduate students is to introduce these students to social, philosophical and ethical issues surrounding agents and multiagent systems, as a means of going beyond basic computer science aspects. Our goal was also to engage students in active learning[3] and teach them about challenges of defining agents and multiagents as a field and to the core issues of what it means to be intelligent. The aim was also to tie the material in class to a broader context and (hopefully) make connections to other classes/areas of interest, e.g. philosophy. There were two separate activities undertaken in this context, as discussed below.

4.0.3 **Trial of a Robot**

A series of two-three lectures focused on the trial of a robot to bring up fundamental philosophical arguments surrounding agents. Here we showed students the film clip of the “The trial of Commander Data” from “Star Trek: The next generation.” Commander Data is an Android who is put on trial to determine whether or not it has rights. In particular, if the ruling of the trial is that Commander Data has no rights, then he would be immediately dismantled. If it has rights, then he would be immediately dismantled — with a slim chance of being put back together. This episode was written by the science fiction writer Melinda Snodgrass.

The trial took place in two separate lectures. In the first lecture, students were shown the first half of the episode that chronicles the events leading up to Data’s trial. We then divided the class into four teams and gave the teams time to strategize/coordinate. Each team was assigned to argue either the pro or con of one of these two issues:

- Is commander Data intelligent? self-aware? sentient?
- Does Commander Data have rights? If Data creates art, who owns it? If Data kills someone, who is responsible?

We provided the students with a list of readings, supporting both pro and con positions. On one side were Turing’s “Computing Machinery and Intelligence” [16] and Asimov’s essays [1] and on the other were readings such as “Can a computer have a mind” [11] and Searle[14]. To prepare for the trial, students had to do the following homework:
As homework, you need to read 2 readings from the list provided below, at least one must be non-Asimov. Then, write a short (roughly half a page) list of three supporting arguments for the position you have been assigned. Your arguments can either be direct support or refutations of likely counter-arguments that the opposing team will make. For each of your points, please provide some support based on a reading. 1 point of extra credit can be earned on this assignment if you bring in an additional credible source to support your argument and provide a citation.

In the second lecture on this topic, students re-enacted the trial in class. During the trial, students had to speak up in support of their position based on their writeup submitted as part of their assignment. As instructors, we were not completely sure how this trial would work out. Our observation was that students were really passionate in support of their positions and had researched many extra sources. Several new and innovative arguments were brought to the floor, e.g. one student brought up the National historic preservation act to argue that Commander Data could not be dismantled because it was a historic Engineering artifact. During the last iterations of this class, we had a third lecture where the author of this episode, Melinda Snodgrass, visited our class and gave an invited lecture. The lecture was of general interest, and so other university audience was also invited. Starting with providing students the underlying motivation for this particular episode, the Dred Scott Supreme Court decision in the United States (based on slavery)\(^1\), the author went on to describe the role of robots in science fiction. She explained how she had investigated different human characteristics using robots, e.g. what would it mean for a robot to be leading a group of humans and what leadership entails for a robot. This contributed to both the philosophical fundamentals, but also the sense of awe about intelligent agents to inspire students to further continue studying in our field.

4.0.4 Futuristic Concerns about Agents

Science fiction writers, mass media reporters and some non-fiction writers have presented stories and scenarios that express concerns about future intelligent agents and multiagent systems and the harm they may potentially cause humans\(^1,9\). For example, in some science fiction movies or stories, we see agents/robots developing their own agenda and ultimately even killing humans. In others, agents are specifically designed to intrude on people’s privacy. In still others, agents may malfunction and create problems. In other words, science fiction writers have been worried that agents may cause us harm in any one of a number of ways: emotional, financial, physical, societal and so on.

The traditional view of AI education has been to ignore such concerns, in part because addressing such concerns is sometimes outside the scope of AI researchers’ expertise, in part because some of these concerns appear too futuristic, and potentially due to other reasons. Fortunately, AAAI (Association for Advancement of Artificial Intelligence) has now recognized this concern, and it has recently formed a panel to study and report on this issue. Furthermore, some researchers have already begun focusing on research on the so called “Asimovian intelligent agents”, explicitly inspired by Asimov’s three laws for safety\(^5\).

\(^1\)For the international readers who may be unfamiliar with this case: The Dred Scott decision in 1857 by the United States Supreme Court ruled that people of African heritage brought to the United States as slaves were not and could never be citizens of United States, and had no right to sue.

Our goal however was to bring these concerns to the notice of our students, and more importantly, to understand what mechanism, engineered either at design time, or some via “social law” or societal convention would help address these concerns. To that end, students were again divided into teams. Each team was given an assignment to focus on a movie or a short story where agents caused harm. Specifically, the assignment suggested to first identify a design approach based on techniques learned in class, either BDI, or decision-theoretic or swarm intelligence that could be used to build the agents. Next:

Identify key scenes in the movie/story where the agent causes harm. Using your approach explain how it could give rise to to such a harmful action, if not properly designed. Pick two scenes in particular and answer two questions: (i) explain what might cause the problem shown to arise. (ii) how realistic is the problem described.

The next step was crucial: identifying potential design decisions, social conventions, laws, that would significantly reduce or eliminate the potential for such harm. The key was to allow agents to exist functionally; in particular, very strong restrictions could completely eliminate agents’ usefulness.

The result of this assignment were projects that outlined design of agents in different popular science fiction movies and stories. A range of solutions have been proposed. In one case agents’ emotions were held responsible, and thus the idea of providing future agents with emotions was shown to be of concern. In another case, a student group analyzed Asimov’s three laws, pointing out how these laws would need to be specialized before we delve further into the “Asimovian agents”.

5. FEEDBACK AND LESSONS LEARNED

Results from student feedback on the courses have been encouraging. Students provided overwhelmingly positive response, not only about the course material but also about the teaching technique of using science fiction. More specifically:

- While exact numeric evaluation scores from students on the course may be difficult to evaluate in the abstract (without having an appropriate baseline), it is useful to note that the scores received have been among the highest that the instructors have received among all courses they have taught.
- We provided students with additional questions to evaluate the role of science fiction: 16/20 students who provided feedback thought that the science fiction really added value to the material taught. Students commented that without the science fiction they would not have taken the course.
- At least some students (from our classes focused on non-majors) have suggested that they will change or have changed their majors to computer science; students from the “welcome seminar” have continued to correspond via email querying about agents and multiagent systems.
- Half-a-dozen students from the upper division computer science class have joined AI research labs to pursue research in agents and multiagent systems, either as undergraduate researchers, PhD students, or research programmers.
- A few of the students also immediately followed up the courses to take more in-depth courses in computer science.
Given the success of our courses, it is useful to step back and understand some of the key factors that in our view contributed to this success. These include:

- We were teaching material for which there was no clear textbook, bringing in cutting edge research material into our courses, and simultaneously using new teaching tools. This required significant amounts of planning, even much more so than traditional courses; we were fortunate enough to have been forced into this planning a year in advance partly by our faculty colleagues who engaged us in lively discussions.
- Figuring out the right science fiction material to use given our criteria proved to be extremely difficult. If the trend of using science fiction in AI were to be carried forward, it would help to build up a collaborative database of science fiction materials utilized.
- Finally, colleagues in AI and in particular agents and multiagent systems from around the world were extremely supportive of this effort, providing both encouragement and pointers to relevant materials.

6. SUMMARY AND RELATED COURSEWORK

It is now time to step up to the challenge of introducing agents and multiagent systems to undergraduate students. To that end, we have used science fiction and games in the classroom as a means to bridge the gap in students’ understanding of agents and multiagent systems, and to generate excitement about our science. Our mission was not only to teach students fundamental concepts in multiagents and excite them about AI and computing in general, but also to provide students with a broader view of the social and cultural context of the development of intelligent agents, including a discussion about the ethical, philosophical and societal issues relating to this work. To that end, since the Fall of 2006, we have taught several courses at two different universities, both to majors and non-majors.

This paper outlined the courses taught using this framework, provided an overview of our classroom teaching techniques in using science fiction and classroom games, and discussed some of the lectures in more detail as exemplars. We discussed the overwhelmingly positive student response and provided concrete examples of students turning to multiagents research as a result as well as to changing their majors to computer science.

There are other courses offered in other universities that use science fiction as a way of introducing science in general. Prof. Barry Luokkala of Carnegie Mellon University, Teaching Professor in the Physics department, uses science fiction for introducing science. Our mission was not only to teach students fundamental concepts in multiagents and excite them about AI and computing in general, but also to provide students with a broader view of the social and cultural context of the development of intelligent agents, including a discussion about the ethical, philosophical and societal issues relating to this work. To that end, since the Fall of 2006, we have taught several courses at two different universities, both to majors and non-majors.

Other similarly offered courses include:

- NIH’s program called “Science in the Cinema”
  http://science.education.nih.gov/cinema

- the American Chemical Society’s similar program:
  http://www.scalacs.org/ScienceCinema/

In particular, those programs also use film and world leading scientists commenting on the films to educate students and the general public. Bringing some of these approaches to teaching agents and multiagent systems has resulted in enthusiastic and strongly positive student response.

7. REFERENCES


2. PART II: Multiagent interactions
   (a) CHAPTER 8: Agent Modeling
   (b) CHAPTER 9: Emergence, Biologically-inspired Multiagent Systems and Swarm Intelligence
   (c) CHAPTER 10: Multiagent Teamwork: BDI and POMDP approaches
   (d) CHAPTER 11: Team and Coalition Formation
   (e) CHAPTER 12: Agent Networks: Distributed constraint satisfaction (DisCSP) and Distributed Constraint Optimization (DCOPs)

3. PART III: Agent-human interactions
   (a) CHAPTER 13: Agent-Human Interactions
The SEREBRO Project:  
Fostering Creativity through Collaboration and Rewards

Rose Gamble  
gamble@utulsa.edu  
Sandip Sen  
sandip-sen@utulsa.edu  
Bradley Brummel  
bradley-brummel@utulsa.edu

Frank Grove  
dean-grove@utulsa.edu  
Noah Jorgenson  
noah-jorgenson@utulsa.edu  
David Baker  
david-baker@utulsa.edu

Dan Guernsey  
dan-guernsey@utulsa.edu  
Daniel Hampton  
daniel-hampton@utulsa.edu  
Jordan Hughes  
jordan-hughes@utulsa.edu

University of Tulsa  
800 South Tucker Drive  
Tulsa, OK 74104

ABSTRACT
Software Engineering is a highly creative endeavor that challenges Computer Science (CS) students to establish an innovative vision and to craft an outstanding product. Curriculum standards for CS education typically lack creative approaches to Software Engineering, focusing on technological solutions rather than innovative design. Accountability for and contribution to creative initiatives are therefore not part of grading methods in typical Software Engineering courses. In this paper, we introduce a unique framework to foster creativity within an asynchronous, interactive, and graphical environment that tracks the team’s product understanding through the phases of the Rational Unified Process for software engineering. We incorporate a layered, multiagent system to apportion rewards for creative contributions that correspond to theories of creativity based on external motivation.

Keywords
Multiagent Systems, education, learning

1. INTRODUCTION
Creativity is defined as the ability to produce work that is novel, high quality, and appropriate [34]. Researchers generally agree that creativity is influenced by a number of factors, such as personality, motivational, environment, skills, and knowledge [37]. Recent theories posit that creativity is a decision. According to the investment theory of creativity [36], individuals decide whether to use their resources to generate new ideas, evaluate those ideas, and then sell those ideas to others. The SEREBRO project targets the malleable factors that affect creativity. Incentives, motivation and rewards for creativity are typically lacking in undergraduate Computer Science (CS) education due to existing curriculum guidelines that focus on technological solutions rather than innovative designs. The SEREBRO project has three objectives for fostering and rewarding creativity within the pedagogical environment of a year-long Software Engineering Projects class at the University of Tulsa:

- Build a graphical, asynchronously communicated, idea network to interconnect ideas and support creativity via a social networking platform.
- Tailor idea network usage to explicit phases of software development to encourage creativity toward project milestones.
- Investigate mechanisms that assess idea proliferation and propagation to reward creativity.

This project synergizes expert knowledge and experience in software engineering, multiagent systems and learning, and industrial/organizational psychology. In this paper, we present our novel framework for developing a multi-faceted educational software system called SEREBRO (Software Engineering REwards for BRainstorming Online). This system forms the foundation of the project to understand how creativity in software development can be enhanced through technology and reward.

2. BACKGROUND
In this section, we overview the different aspects related to the SEREBRO project.
2.1 The Creative Process

Creativity has been studied from a variety of perspectives. One of the most common distinctions has been between the creative process and individual differences in creative behavior. Our project focuses on enhancing the creative process by applying various aspects of two theories of creativity. Sternberg and Lubart’s [36] investment theory of creativity states that in order to be creative, individuals must be willing and able to “buy low and sell high.” In other words, creative individuals pursue unpopular or unknown ideas with potential and then persist until they convince others of the idea’s value. The underlying idea of this theory is that “creativity is in large part a decision that anyone can make but that few people actually do make because they find the costs to be too high” [35]. In this theory, creativity requires a confluence among six critical resources: intellectual skills, domain knowledge, thinking styles, personality, motivation, and environment.

Amabile’s componential model of creativity [3, 4, 5] states that there are three critical components necessary for creativity: domain-relevant skills, creativity-relevant processes, and intrinsic task motivation. Domain-relevant skills include a minimal level of domain-specific talent and expertise. Creativity-relevant processes involve the ability to deal with complexity and break through one’s mental set when problem solving, an understanding of methods for producing novel ideas, and a high-energy, focused work style. Intrinsic task motivation refers to “the motivation to engage in an activity primarily for its own sake, because the individual perceives the activity as interesting, involving, satisfying, or personally challenging” [7]. Individuals will be intrinsically motivated to the extent that the task is interesting, they have autonomy, they feel competent to perform the task, and they feel a sense of self-determination and control. On the other hand, external motivation is “the motivation to engage in an activity primarily in order to meet some goal external to the work itself, such as attaining an expected reward, winning a competition, or meeting some requirement” [7].

This model posits that intrinsic motivation enhances creativity, while extrinsic motivation can either undermine or support creativity. Specifically, research has found that creativity suffers when task autonomy is reduced [14], competition is introduced [2], or a performance evaluation is expected [1]. Amabile and colleagues term these “non-synergistic extrinsic motivations” because they undermine one’s self-determination and competence. However, extrinsic motivation that reinforces one’s self-determination and competence may support creativity. Rewards and recognition for creative ideas, clear task goals, and performance feedback that confirms competence and provides guidance on how to improve should enhance creativity. Amabile and colleagues refer to these external motivators as “synergistic extrinsic motivations” because they can combine with intrinsic motivation to foster creativity. The creative process involves problem presentation, idea generation, preparation, idea validation, and idea communication, and is generally thought to occur in a non-linear fashion [3]. As previously mentioned, creativity requires both novelty and appropriateness [3, 34]. Novelty is critical during the early stages of the creative process, while appropriateness is more important during the later stages [4, 5]. Intrinsic motivation is especially important for fostering novelty, and synergistic extrinsic motivation helps individuals persist and focus on the upcoming evaluation. Thus, intrinsic motivation is especially important in the early stages of the creative process, while synergistic extrinsic motivation is particularly important during the later stages.

Because many creative activities take place in a social context [11], it is essential to consider how the group or team can influence the creative process. One model of team innovation states that team creativity and innovation will be influenced by group task characteristics and group knowledge diversity and skills [41]. Intrinsic motivation should be highest when the group task is a whole task, creates varied task demands, and allows autonomy, learning opportunities, and social interaction opportunities. These task characteristics are consistent with socio-technical systems theory, which emphasizes the need for autonomous teams to optimize the social subsystem with the technical subsystem [8]. Both group task characteristics and group knowledge, diversity, and skills should affect group creativity and innovation via integrating group processes that support these goals. Specifically, group processes should clarify and ensure commitment to the group’s goal, allow participation when making decisions, manage conflict effectively (i.e., constructive task-related controversy), support innovation, and create a sense of safety for creativity [41].

2.2 Creativity and Software Engineering

It is generally agreed that in order to truly support creativity through the use of a software tool, the tool should accommodate all skill levels and seamlessly interface with other tools the user might employ in their creative endeavor [26, 29]. Interface usability is an important aspect of creativity support tools (CSTs) because the tool should not hamper creativity [18]. Production of these tools has not reached the quality needed for use in the classroom. We believe, however, that collaborative problem solving activities for course projects can benefit from CSTs. These same activities are also ideal for studying creativity processes as students with overlapping knowledge levels, expertise, and capabilities collaborate in a supportive environment [9]. While creativity has been widely studied in the social science literature, CSTs are not yet mature for IT applications. Some key desirable features for CSTs as identified by researchers are [6, 29]:

- exploration and collaboration with open interchange
- simple, attractive, and non-intrusive, yet powerful
- flexible, with support for different learning, designing, and thinking styles
- allow asynchronous and non-linear work flow.

Researchers studying Computer Supported Collaborative Work [13, 27] have focused on the distributed, social aspects of creative activity [11, 19] resulting in more discourse on CSTs. Several brainstorming tools exist for eliciting an improved return-on-investment. Three categories of tools are prevalent: idea prompt generators for associational thinking [28], mind mappers for visual thinking [42, 25, 16], and information organizers for organizational thinking [15, 24]. Few of these tools combine the visual and organizational aspects of brainstorming. None include a concept of rewards or personnel evaluation.
2.3 Software Development Process

A standard development process structures the development of non-trivial software from the first set of requirements to product delivery. For software engineering classes, we use the Rational Unified Process (RUP) [30] which provides relevant content to empower students with immediate process knowledge though they have limited experience. Figure 1 is a modified version of RUP from [30].

The RUP is an iterative process with four phases: Inception, Elaboration, Construction, and Transition. An iteration achieves particular tasks and milestones incrementally, breaking the software development across smaller, more manageable disciplines for each phase. Thus, each iteration contains a portion of the development effort by gathering and clarifying requirements, establishing design and architecture models, implementing executable portions of the product, and testing implemented code along with code integration.

The Inception Phase clarifies the interpretation of the customer’s requirements, explores technology solutions to reduce product development time and increase quality and performance, and generates an initial set of artifacts, such as a vision document, an architecture, and a throw-away prototype. If done properly, vast amounts of intellect and creativity can be communicated, discussed, and decided upon before leaving the inception phase. Milestones are reached upon receiving positive feedback from the customer and, in the case of a class, a good grade from the instructor. Though mistakes can lead to the wrong path, the iterative approach embedded within the RUP keeps them contained and identified early in the development process. The Elaboration phase refines initial artifacts and moves the team toward the software design. Milestones of Elaboration include creating the experimental arrangement of architecture entities and the building of an executable prototype. The Construction phase has multiple milestones for coding and testing, though some requirements gathering and design interpretations occur. In this phase, creativity is often associated with the use of technology, efficient coding and reuse, and repeated contact with the customer. Transition is the process of delivering the system in alpha, beta, and final versions to the customer site. Creative development of the user’s manual is part of this phase. The RUP is a feed forward approach because considerations are given to all disciplines of the product in all phases. Thus, creative activity can be restarted and refocused within each iteration. Though versioning is included in tools that organize software projects according to the RUP, creative brainstorming and idea organization are not available within these tools.

2.4 Problem Structures in MultiAgent Systems

Various network models have been proposed to model agent interactions in a multiagent system [17, 33, 40]. We are particularly interested in a distributed goal representation framework that allows effective coordination between multiple agents [23]. The Generalized Partial Global Planning (GPGP) and its associated TAEMS task network representation have been developed as a domain-independent on-line coordination framework to facilitate cooperation by small teams of intelligent agents. The framework develops a distributed solution to a global optimization problem by decomposing the global problem into a set of dynamically evolving local optimization problems assigned to individual agents. Initially agents possess only local views of their assigned subtasks and their interrelationships stored in the TAEMS representation, which adds quantitative information and sequencing constraints to AND/OR goal trees. GPGP provides information gathering and coordination to develop more global views and make commitments to facilitate coordination. While GPGP/TAEMS is a complex, elaborate framework, for our purpose, the relevant components are the use of a set of hard and soft relationships between tasks to represent various non-local effects. An example hard constraint is enables which denote that completion of a sub-task is necessary before another sub-task can be processed. Facilitates is a soft relationship signifying results from a sub-task may be useful for processing another task. We plan to adapt and expand these features from the GPGP/TAEMS framework to provide the set of activities available for creative collaboration in SEREBRO to codify the links between ideas.

A key component of the SEREBRO framework is the feedback of external reinforcement over various internal and intermediate actions, proposals, etc. The postings, suggestions, comments of the users are used to develop a multi-connected, feed-forward network structure. While various learning schemes can be used to feedback external reward over internal nodes in a network, e.g., backpropagation scheme in neural network structures [31], we believe reinforcement learning techniques are most appropriate for addressing the challenge of effective reward distribution [20, 38]. Payoff distribution among agents in a coalition has received significant attention in multiagent systems research [21, 32, 39]. Solution concepts like the core, the Kernel, etc., have been proposed to generate payoff sharing that guarantees stable coalition structures. The current paper does not directly use these techniques because we assume that a stable group of users are using the system. However, we plan to use aspects of fair payoff distributions from the above-mentioned research to create more fair reward structures based on creative participation and contribution by the users towards effective design solutions.

3. THE SEREBRO TOOL

The main challenge behind recognizing and rewarding creative contribution by team members in a Software Engineering project is to devise a framework that captures and relates ideas as they are generated, guides the creative design pro-
cess toward the project artifacts and milestones, and rewards those who contribute to the project’s creative elements. Our core notion of the creative process is an idea. An idea can be a belief statement, a problem solving approach, a solution to a problem, or a discussion related to any idea type. Idea-related activities include:

**Brainstorming**: asserting initial ideas, vision, and designs without past support;

**Spinning**: connecting supporting ideas to support progress towards milestone;

**Pruning**: disallowing further spinning of an idea and forcing the pursuit of other paths;

**Finalizing**: declaring an idea as a decision-action node to signify a definite team agreement to complete an iteration goal.

The partitioning of nodes into the above roles aids the association of creativity metrics, and reward distribution. Brainstorming allows for asynchronous, spontaneous input. Spinning produces pathways for creative expression from initial concept to ideas that agree and extend the concept or ideas that disagree and offer contrasting input. Pruning is a joint decision of contributors to stop (perhaps temporarily) a non-productive pathway so that focus can be put toward other ideas. Finalizing is the collective decision or result that stems from multiple ideas. Finalized nodes move the team toward achieving a milestone associated with a RUP iteration or completed phase.

These nodes are linked to each other in a feed-forward manner capturing the time-line of activities to form an idea-network. Since our approach applies links to ideas instead of users, it allows feedback to be given on a per-idea basis. It also allows measurements to be taken based on the links between ideas. The social activity that centers on ideas is associated with actual users for the purpose of rewarding the correct person. Thus, analysis can be performed on the ideas much the same way that it is performed on the social network of people [22, 10] to determine how creativity enhances the team software development experience.

Our web-based application, called SEREBRO 1.0, is an idea generation tool that incorporates graphical and textual idea expression, idea connectivity, and agree/disagree capability. It is used as part of a 2-semester course sequence in software engineering. Its pedagogical goal for undergraduate software development teams is to enhance students’ knowledge of software engineering principles and techniques through participation in creative group activities. Personal responsibility and satisfaction combined with goal achievement are stressed. The plan is to couple motivational methods with reward systems that overlay dynamic idea generation dedicated to fulfilling the requirements of a software project.

Figure 2 provides an overview of the SEREBRO tool. The upper right section shows the Forum in which topics are posted across the RUP. The lower right section shows the Graph View of an idea network. Agree nodes are green triangle. Orange triangles are disagree nodes. Yellow pentagons are finalized ideas. Each node has a corresponding post entered in a thread-based environment that reflects idea connectivity, as seen on the left half of the figure. The left portion of the figure labeled as Post View reflects a sample of that thread-based environment. Within the Post View section, the lower left portion of this section reflects the finalization of a node in which tags indicate those ideas that contributed to the final content, while the lower right portion of this section shows the stop procedure used for pruning a thread of discussion.

### 4. MULTIAGENT REWARD FRAMEWORK

To properly facilitate and allocate the internal and external reward distribution, we developed a multiagent framework. The framework is designed to allow a fair and responsive reward distribution by assigning a user-agent to each individual participating in a SEREBRO project, an agent for each phase of the RUP, and an arbiter agent to finalize and distribute rewards. These agents must negotiate a mutually acceptable distribution of internally or externally generated rewards. Examples of external reward includes those received from instructor evaluation, user-satisfaction of delivered product, etc. Examples of internal reward includes those received based on contribution and influence of individual participants in the group problem solving process.

In order to determine the importance of each node within the SEREBRO framework, these agents will distribute points to other agents that the user has responded to. At pre-specified times in the classroom agenda, the agent’s aggregate points are passed back over the network to the phase agents. Each phase agents then computes a weight $W_p$ vector with weights $w_{ui}$ for each user participating in that phase. After each phase in the RUP is completed, the phase-agent submit $W_p$ to the arbiter agent, who distributes the rewards. In the final phase, the phase agents negotiate over the final weight vector $W_f$. Next we present further details of the multiagent framework, negotiation protocol, and the point distribution system.

The agent based reward distribution framework is designed to achieve a fair and accurate distribution of rewards by the SEREBRO system. In the future we expect the reward system to handle both external feedbacks, from users of the product or service generated by using the system, as well as internal feedback, based on activities of the participants involved in generating the product using the SEREBRO system. In this paper, we focus on the generation and distribution of rewards based only on internal feedback.

Each user is assigned a user-agent $i$. Each user-agent $i$ has a point repository and must adhere to the point distribution protocol. The protocol distributes points based on the participation of the users and the estimated overall utility of an idea. Our goal in designing the reward distribution scheme was to handsomely reward ideas that spawn large discussions while being less enthusiastic in rewarding ideas that do not generate fruitful responses. Hence, the topology and links in the idea network play a pivotal role in the reward amount distributed to an idea node. The goals identified above are supported as follows:

**Agree with a Node**: When a user agrees to a node, $x$ points are created and passed to the node being replied to. If the user is agreeing with himself/herself, no points are distributed.

**Disagree with a Node**: When a user disagrees with a node, the agent that created the node being replied with is charged $\frac{x}{2}$ points.
Agent Receives $x$ Points from Reply: When a user’s node $a$ is replied to with an agree node $b$, the user-agent who posted $a$ will receive $x$ points. If the node $a$ is also a reply, the user-agent must pass $\frac{x}{2}$ to the node $c$ to which it replied.

Finalized Node: The finalized node allocates $k \times x$ points to the node it is replying, and $x$ points to each node that is tagged with it.

This protocol efficiently allocates points to the users that participate the most and whose participation is most appreciated. Over time the points will accumulate which also allows instructors to observe the progress of threads and individual users over time. At an arbitrary time selected by the instructor, rewards will be allocated. At this time, each user-agent submits their points to the phase agents where the points originated. The phase-agents then simply pass along the points, so that the arbiter may distribute $N$ rewards to the top $N$ users. These rewards may occur during the end of a RUP phase, as employed by the instructor, or at the demonstration of a prototype. The process that determines the final reward must take into account the points accumulated in all four phases.

Based on the communication between users, each phase agent $p$ computes a point distribution, $W_p$, containing the points each student accumulated within that agent’s respective RUP phase. These values are weighted by the satisfaction vector $S_i$. In order to properly determine the reward distribution, the phase-agents must negotiate an outcome that maximizes their satisfaction vector $S_i$ for each user $i$. This satisfaction vector magnitude is a number in the range $[0,1]$.

$S_i$ ensures the relevance of outliers in the point distribution. Since point distributions from each phase vary, no phase-agent’s $S_i$ magnitude is 1. In the improbable case when these point distributions are equal there is no need for negotiation. These outliers created by the difference in distributions are addressed through a bidding process. This process is important because these outliers can be extreme. For instance if a user had low points in an early phase but exceeded all others in later phases, the user’s contributions must be considered carefully. This bidding process begins when each agent submits a bid to the arbiter agent equal to its point distribution $W_p$.

The bids are received by the arbiter which takes the average $w_a$ and returns this point distribution to each phase agent. The phase-agents then make incremental changes to the point values of the averaged distribution to maximize the sum of $S_i$ and resubmits the new point distribution as a bid to the arbiter agent. This process continues until the arbiter agents detects convergence, i.e., the difference between the bids and the average falls below a threshold, or after...
a maximum number of bidding sessions. Once the arbiter has ceased bidding rewards are allocated based on the final average point distribution.

5. SEREBRO USE

5.1 Software Projects I

Software Projects I is the first class in the two-course software engineering sequence that is required for seniors in Computer Science. Multiple software challenges are explored by change teams of students throughout the semester. Toward the end of the semester, final teams are chosen and assigned projects [12]. Initial ideas for product name, vision, and functional requirements are needed for the team to complete the semester final. To test the functionality and usage outcomes, we evaluated the first prototype, SEREBRO 1.0, against SEREBRO Lite, a thread-based tool that only showed connected textual posts without the ability to agree or disagree with a post. Student teams were randomly assigned to either SEREBRO 1.0 or SEREBRO Lite 1.0. We focused on evaluating:

- Satisfaction with the tool
- Their feeling toward creative expression
- Their perceived benefits to the team and to the progress of the project

Among those students using SEREBRO 1.0, the response to a series of questions indicated the overall feelings for SEREBRO were positive. Those surveyed were satisfied with the usefulness of SEREBRO for idea communication and organization. Most indicated a positive feeling toward creative expression. These teams had the visual benefit of seeing a response to their ideas, as well as having an understanding of the teams perception of an idea according to its graphical connectivity. Many of the features currently implemented in SEREBRO were expressed as being highly desirable by the reviewed users. The thread based connectivity of ideas, the agree/disagree options for idea connectivity, graphical display of idea nodes and the tag-based system to meta-connect ideas were all preferred by nearly 80 percent of those users. In particular, the functionality for uploading files for sharing and email notification upon node creation were extremely well received, with a nearly unanimous 90 percent indicating the usefulness of these features.

<table>
<thead>
<tr>
<th>Percent rated functionality as useful</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idea communication</td>
</tr>
<tr>
<td>Idea organization</td>
</tr>
<tr>
<td>Thread based connectivity of ideas</td>
</tr>
<tr>
<td>Upload of files for sharing</td>
</tr>
<tr>
<td>Version control for uploaded files</td>
</tr>
<tr>
<td>Agree/disagree options for idea connectivity</td>
</tr>
<tr>
<td>Graphical display of idea nodes</td>
</tr>
<tr>
<td>Tag-based system to meta-connect ideas</td>
</tr>
<tr>
<td>Email upon node creation</td>
</tr>
</tbody>
</table>

Table 1

Examination of the SEREBRO graphs and threads from the fall experiment correlated to the results as we saw a clear distinction between the users on SEREBRO 1.0 and SEREBRO Lite 1.0. While the students on SEREBRO 1.0 were utilizing the agree and disagree options and properly creating new brainstorm nodes when a new idea was being formed, the groups on SEREBRO Lite were noticeably more inactive than the users on SEREBRO 1.0. This was so much the case that one group on SEREBRO Lite hardly used the tool. The groups using the SEREBRO Lite didn’t display as good idea organization, and the spinning of ideas was nearly nonexistent with no real development if an idea was posted. In contrast, the users on SEREBRO 1.0 were more active and in total had more posts with more content, most likely the result of a highly visual graphical view of the nodes and the ability to specifically agree and disagree when discussing ideas. This is supported by the survey results indicating that nearly all the students suggested the features unique to SEREBRO 1.0 were more useful overall. Grading was done prior to and without consulting SEREBRO activity. Thus, grading was solely based on traditional assessment of artifacts turned by teams. Once grading was complete, those teams that used SEREBRO 1.0 had an average of 10 percentage points higher on their final than those that used (or neglected to use) SEREBRO Lite. More assessment is needed to determine if usage correlates to higher quality artifacts because of SEREBRO or if the teams were actively working on their final and SEREBRO was another part of that activity.

5.2 Broader Evaluation and Use

Currently, SEREBRO is undergoing extensive testing as part of the second half of the Software Engineering Projects class sequence. Three non-trivial software projects are under development:

- A chemistry lab creator with automatic scoring and database history
- A game aimed at recruiting junior and senior high school students to computer science
- A Web services management system for responding to crises

Students are separated across six teams with two teams per project. Standard rubrics developed for accreditation purposes and from previous years’ classes are used to grade core artifacts (e.g., documentation, code, and demonstration) for the four ‘builds’ during the semester. The structure of the class coupled with the traditional grading approach allows us to directly compare and contrast quality of the project and productivity of the team with the creativity metrics and reward system that are part of SEREBRO.

In the Inception phase, teams are using SEREBRO to identify additional functional requirements, technology for use in the product, and programming platform. They also develop a prototype to present and be judged by the class for its sufficiency in demonstrating a risky or complex requirement and the flawlessness with which it is scripted and shown. At the end of the phase, SEREBRO will be used to finalize their ideas to move to the Elaboration Phase. The Elaboration Phase should see creative ideas for generating design artifacts such architecture, class, and sequence diagrams. Additional prototype development will take place where the teams can discuss the implementation of various requirements. Another integral part of Elaboration is the understanding of project risks related to technology, experience, and customer interpretation that can cause a project to fail. Idea threads will be created per identified risk for
replied to most frequently and most positively. We believe this will bias our results towards users whose comments are for points to be accumulated. Initial analysis shows that users must respond positively to a user’s comments in order in the discussion may not necessarily score points. Other have already begun to diverge.

existing discussions, we can see that the individual scores found in Table 2. While this is only a small sample from the point distribution scoring on this example can be use a point value of 5.3 Sample reward distribution

As mentioned above, the SEREBRO framework is currently being tested in a Senior Software Project class. We now present a small sample graph from this class to illustrate the point distribution process. This example is demonstrated in Figure 3. In this example users A, B, C, and D are taking part in a discussion related to the project. We use a point value of \( x = 1 \) for this example. The results from the point distribution scoring on this example can be found in Table 2. While this is only a small sample from existing discussions, we can see that the individual scores have already begun to diverge.

From this example, we can see that users who take part in the discussion may not necessarily score points. Other users must respond positively to a user’s comments in order for points to be accumulated. Initial analysis shows that this will bias our results towards users whose comments are replied to most frequently and most positively. We believe our protocol will operate well under the assumption that the creative and productive users are most likely to initiate these threads of discussion.

Although our analysis is preliminary, we believe that the point distribution protocol will perform well in allocation of rewards. As the class moves into the next phases of the RUP, more data will become available to test our protocol against the performance perceived by those taking part in the experiment. At this point we will proceed with more in depth analysis of the point distribution protocols described here and continue with the implementation of the phase-agent’s negotiation protocol.

6. CONCLUSION

In this paper, we introduce the framework for the SEREBRO project to codify creativity theories and metrics within a type of social networking system where ideas are the basis for expression and reward. We show actual screen shots from SEREBRO’s graph and post views, along with the types of ideas expressive and where in the software development process they can be housed. Our multiagent reward system is described with initial reward assignments to project teams. The SEREBRO system brings together complementary expertise of research groups from software engineering, psychology, and multiagent systems to promote creative collaboration and problem solving by CS students in group projects. While initial testing shows performance improvement of students of SEREBRO 1.0 over those that use SEREBRO Lite, much remains to be done to harness the potential of cross-fertilizing ideas, concepts, methodologies, and processes from these diverse research areas. In particular, the interplay of external and internal reward metrics and their redistribution over the idea network produced based on asynchronous group problem solving poses exciting opportunities and challenges for identifying and fostering fruitful, creative user contributions.

Acknowledgement.

This work is supported in part by a National Science Foundation grant IIS-075743.

7. ADDITIONAL AUTHORS

8. REFERENCES


Enhancing Training by Using Agents with a Theory of Mind

Maaike Harbers
Utrecht University
P.O. Box 80.089, Utrecht
The Netherlands
maaike@cs.uu.nl

Karel van den Bosch
TNO Defence, Security & Safety
P.O. Box 23, Soesterberg
The Netherlands
karel.vandenbosch@tno.nl

John-Jules Meyer
Utrecht University
P.O. Box 80.089, Utrecht
The Netherlands
jj@cs.uu.nl

ABSTRACT
Virtual training systems with intelligent agents are often used to prepare people who have to act in incidents or crisis situations. Literature tells that typical human mistakes in incidents and crises involve situations in which people make false assumptions about other people’s knowledge or intentions. To develop a virtual training system in which correctly estimating others’ knowledge and intentions can be trained in particular, we propose to use agents that act on the basis of their own mental concepts, but also on mental states that they attribute to other agents. The first requirement can be realized by using a BDI-based agent programming language, resulting in agents which behavior is based on their goals and beliefs. To make these agents able to attribute mental states to others, they must be extended with a so-called Theory of Mind. In this paper we discuss the possible benefits and uses of agents with a Theory of Mind in virtual training, and discuss the development and implementation of such agents.

Keywords
Virtual training, Theory of Mind, BDI agents.

1. INTRODUCTION
Virtual training systems are often used to train people who are in command during incidents or crisis situations. In an increasing number of training systems, intelligent agents are used to generate the behavior of virtual characters in the training scenarios, saving time and costs. We consider training systems in which one trainee has to interact with one or more of these agents to accomplish a certain task or mission in the scenario. Though simple agent behavior can be accomplished quite well, the development of agents displaying more complex behavior is still to be improved.

Typical human errors in incidents or crises involve situations in which people make false assumptions about other people’s knowledge or intentions. For example, a leading fire-fighter that receives information about a second fire in a building may immediately make an assessment of the new situation, come up with an alternative plan, and redirect part of the team. In the rush of the moment, the leading fire-fighter could unjustly assume that other people nearby, e.g., ambulance personnel, already know about the second fire, unnecessarily exposing them to high risks. Besides similar stories of professionals [10, 28], attributing incorrect knowledge and intentions to others in general is a well described phenomenon in cognitive sciences (e.g. [20, 15]). To provide a system in which correctly estimating others’ knowledge and intentions can be trained, we propose to use agents that act on the basis of actual mental concepts, and are able to attribute mental states to other agents.

The first requirement, agents acting on the basis of mental concepts, can be realized by using a BDI-based agent programming language. The BDI-model is based on folk psychology, which encapsulates the way humans think that they reason [2]. Namely, humans use concepts such as beliefs, goals and intentions to to understand and explain their own and others’ behavior [14]. Accordingly, a BDI agent executes actions based on its beliefs, plans and goals. Several applications have demonstrated that BDI agents are appropriate for modeling virtual characters in games or training systems (e.g. [21, 25], respectively). In virtual training with BDI agents, trainees can practice in correctly interpreting the agents’ behavior and compare their estimated mental concepts to the agents’ actual ones.

To satisfy the second requirement, agents attributing mental states to other agents, the agents have to be extended with a Theory of Mind (ToM). Entities with a ToM attribute mental states such as beliefs, intentions and desires to others in order to better understand, explain, predict or even manipulate others’ behavior. Beliefs about the other’s mental state can be different from beliefs about one’s own. Agents that act on basis of a ToM give trainees the opportunity to experience how their behavior is interpreted by others, and to train on coping with people that make false assumptions about other’s beliefs and goals. As far as the authors know there are currently no agent programming languages providing explicit constructs for the implementation of agents with a ToM.

In this paper, we will explore the possible benefits and uses of agents with a ToM in virtual training, and discuss the development and implementation of such agents. In section 2, we start by providing some background information on ToMs. In section 3, we sketch possible uses of agents with a ToM in virtual training, and therefrom determine implications for the implementation of agents with a ToM. Based on the implied requirements, we introduce an approach for the implementation of agents with a ToM in section 4. We end the paper with a discussion on related research in section 5, and a conclusion and suggestions for future work in section 6.

*This research has been supported by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie).
2. WHAT IS A THEORY OF MIND?

The concept of a Theory of Mind is studied in different research fields. Philosophers and psychologists debate on how a ToM works in humans, developmental psychologists study how children acquire a ToM to obtain more insight into its working, and others study behavior of autistic people, who show deficits in their ToM use. The false-belief task is often used to determine whether someone has a fully developed ToM [27]. To pass the test, the participant has to attribute a false belief to someone else. Furthermore, neuroscientists study neural correlates of ToM use, and biologists discuss about whether primates have a ToM.

All this research aims to improve understanding in the actual working of a ToM in humans or animals. In contrast, in order to develop agents with a ToM it has to be decided how a ToM is designed. Agents with a ToM do not always have to be as similar to humans as possible; the design guidelines for endowing agents with a ToM depend on the purpose for which the ToM will be used. Our aim is to endow agents with a ToM to enrich virtual training. When applied in virtual training agents should behave human-like, but only to a certain extent. Interaction with the agents should prepare trainees for interaction with real humans, but behavior deviating from average human behavior can be used to create interesting learning situations. In conclusion, we inspire our approach of ToM on theories about human ToM, but do not strictly follow them.

The two most prominent accounts of human ToM are the theory-theory (e.g. [6]) and simulation theory (e.g. [11]). According to theory-theorists, a ToM is developed automatically and innately, and instantiated through social interactions. The mental states attributed to others are unobservable, but knowable by intuition or insight. ToMs are not strictly follow them.

3. USES OF TOM-BASED AGENTS

In this section we discuss different uses of agents with a ToM in virtual training systems: providing feedback on trainee behavior, simulating errors due to an incorrect ToM, and supporting the trainee. It should be noted that the last two uses are special cases of the first. Namely, independent of the exact agent model, a trainee should always get feedback on his behavior by interacting with an agent with a ToM. In the last subsection, the implications of the desired uses of agents with a ToM to their implementation are examined.

3.1 Providing feedback on behavior

Agents with a ToM in virtual training should be able to make assumptions about a trainee’s goals, beliefs, and future actions, and based on that determine their own actions. As a result, the trainee gets feedback on its own actions in the form of behavior of other agents. For instance, in the introduction we gave an example of a leading fire-fighter trainee that while handling an incident was challenged with the information about a second fire. Based on the trainee’s behavior, which was redirecting the firefighters, the firefighter agents in his team might be able to derive the trainee’s goals, e.g. splitting the team and fighting both fires simultaneously. Besides, because the trainee did not give any commands about warning bystanders, the agents might assume that the trainee already took care of that. Hence, the agents would not warn people nearby, but instead they could e.g. proactively start to divide the team in two teams. From the agents’ behavior, the trainee make the inference that the others correctly derived the fire attack plan from his behavior, but that they misinterpreted his behavior concerning the bystanders.

Besides feedback through agent behavior, a trainee could also receive feedback on its behavior by ToM-based agents giving explanations about their behavior. In earlier work we have introduced a methodology for developing self-explaining agents [12], which also used BDI agents. According to this approach, self-explaining agents create a log in which, for each action, they store the goals and beliefs that brought about that action. After a training session is over, the agents can explain actions by revealing the goals and beliefs that were responsible for them. To make ToM-based agents self-explaining, not only their own beliefs and goals underlying an action should be stored, but also the attributed beliefs and goals which influenced the choice for that action. The explanations derived from such logs would thus reveal how a ToM-based agent interpreted the trainee’s behavior.

To summarize, agent behavior based on a ToM of the trainee and explanations about such behavior give the trainee insight into how his behavior is interpreted by others. Such insight aims to make the trainee aware of the effects of his actions on others, and let him prevent possible misinterpretations of his behavior. Moreover, by receiving explanations about agents’ behavior, the trainee can check whether his own interpretations of their behavior were correct.

3.2 Simulating errors due to incorrect ToMs

As mentioned in the introduction, incorrect (use of) ToMs is a well described phenomenon in the cognitive sciences. A lot of research demonstrates the human tendency to impute one’s own knowledge to others (Nickerson gives an extensive overview [20]). Although this serves them well in general, they often do so uncritically and assume erroneously that other people have the same knowledge as they have. Sometimes the mechanism thus yields incorrect ToMs.

Keysar et al’s research suggests human limits on the effective deployment of theory of mind [15]. They describe one of their experiment as follows: "A person who played the role of director in a communication game instructed a participant to move certain objects around in a grid. Before receiving instructions, participants hid an object in a bag, such that they but not the director would know its identity. Occasionally, the descriptions that the director used to refer to a mutually-visible object more closely matched the identity of the object hidden in the bag. Although they clearly knew that the director did not know the identity of the hidden object, they often took it as the referent of the director’s description, sometimes even attempting to comply with the instruction by actually moving the bag itself. In a second experiment this occurred even when the participants believed that the director had a false belief about the identity of the hidden object, i.e. that she thought that a different object was in the bag." The results show a stark dissociation be-
between the ability to reflectively distinguish one’s own beliefs from others’, and the routine deployment of this ability in interpreting the actions of others. Some adult subjects could not correctly reason in a practical situation about another person’s lack of knowledge.

Hedden and Zhang [13] conducted experiments in which people were challenged to use a theory of mind in a sequence of dyadic games. Players generally began with first order reasoning (my co-player knows p), and only some of the players started to use second order reasoning (my co-player does not know that I know that p). Møl et al [19] also found that only few people deploy second order ToM in a task where reasoning about others was advantageous. The skill to deploy second order ToM however can be essential in domains such as crisis management and firefighting.

In a training situation, agents could unjustly assume that the trainee has certain knowledge about the situation. For instance, a fire-fighter agent, extinguishing a fire in a building with its colleagues, observes that there is a victim. The agent communicates its observation to its team members, but not to the commander, which is played by the trainee. The agents might impute their own knowledge to the trainee, i.e. they think the trainee also knows about the victim. Consequently, the agents think that it is not necessary to communicate the information to the trainee, and independently start to take actions to take care of the victim. Though the trainee does not know about the victim, he could derive from the agents’ actions that something has happened. For example, he notices that there is not going any water through the fire hoses. In such a case, the trainee could contact the fire-fighters to ask why they are not yet extinguishing the fire. In reaction, the fire-fighters can explain their behavior by their ToM of the trainee, e.g. I thought you knew there was a victim, therefore I started taking care of the victim.

Another example in which the trainee can practice with agents with a limited ToM is that agents have wrong expectations about each other’s tasks. If an agent expects that something is not a goal of the trainee because it thinks the goal does not belong to the trainee’s tasks, it might adopt that goal itself. Then, unnecessarily, two players would try to achieve the same goal. The other way around can even be worse, if an agent unjustly thinks that something is the trainee’s responsibility, the task might not be performed at all. The trainee is challenged to detect these incorrect expectations on the bases of other agents’ actions, and gets the opportunity to deal with such situations.

In conclusion, agents with a limited ToM can be used to create challenging training situations by making realistic errors. Namely, agents with an incorrect ToM about the trainee will not perform optimal behavior, and the trainee is challenged to detect such errors as early as possible and overcome possible problems.

3.3 Supporting the trainee

The goal of a training scenario is to engage a trainee in intended learning situations. Sometimes, this causes a tension between freedom of the trainee and control over the events in the scenario. The trainee’s freedom increases if it can perform a wide range of actions to which the virtual environment reacts in a believable way, from which he might learn a lot. However, too much freedom endangers the continuation of the storyline of the training scenario in a desired way. For example, if a trainee makes bad decisions at an early stage of a session, the situation might quickly walk out of hand and the session be over soon. If the trainee would have reacted more adequately in the beginning, he would have encountered much more learning opportunities.

Agents with a ToM can be used to exert some control over the storyline by supporting the trainee if he makes errors or fails to take actions that are crucial to the continuation of the storyline. Because the agents’ support actions are based on their ToMs and not just artificial interventions, ToM-based behavior is a natural way to balance user freedom and story control. For instance, an agent with beliefs about the current situation and a ToM containing information about the trainee’s tasks, i.e. his goals, can simulate a reasoning process with its own beliefs and the attributed goals. The outcome of the process shows what the agent would do itself in the trainee’s place. If that differs from the trainee’s actual actions, the agent can redirect the trainee by supporting behavior such as correcting the him or taking over his tasks.

Most research to finding a balanced combination of user freedom and control over a storyline has been done in the interactive storytelling community, where the problem is called the narrative paradox [17]. A common solution to the narrative paradox is the use of a director or manager acting ‘behind the scenes’ (e.g. [18, 23]). Its task is to make sure that the scenario is carried out according to predefined constraints, while preserving realism. Generally, story guidance in these approaches happens at certain points of the scenario and leaves the player free space to interact in the remaining time.

The use of agents with a ToM to exert control over a scenario does not contradict approaches with a director agent, instead, both could complement each other. Namely, in some occasions you might want to support the trainee, e.g. if he is performing bad, but in other cases you do not, e.g. to let him experience the consequences of his actions. The agents acting in the training scenario have knowledge about the domain, and know how to support the trainee in such a way that the incident will be solved. A director agent has didactical knowledge and knows on which occasions you do or do not want to support the trainee. The particular design of ToM-based agents facilitates directing them; the director agent can command them to either use or not use their ToMs. The director agent could also control the simulation of mistakes due to incorrect ToMs as described in the previous section by prescribing when agents have to purposely make mistakes and when not.

The advantage a combined ToM-based agents and director approach is that it facilitates direction by a director agent. Moreover, not only the observable behavior of the agents remains realistic, but also the reasoning steps that generated it. The agents’ reasoning processes are not interrupted, but changed in a plausible manner. Consequently, agents’ self-explanation would still deliver useful explanations, even if their behavior was redirected. We have discussed the ideas in this paragraph more extensively in [24].

3.4 Implications for implementing a ToM

By interacting with agents with a ToM and studying their explanations, a trainee obtains indirect and direct feedback on his behavior, respectively. In our approach to self-explaining agents, we argue that the explanation of behavior should be connected to its generation [12]. Thus, intentional self-explaining agents should not only behave as if they were...
intentional, but act on the basis of actual goals and beliefs. As a result, the self-explaining agents can be best implemented in a BDI-based agent programming language. Similarly, self-explaining agents with a ToM should not only behave as if they had a ToM, but actual attributed mental concepts should play a role in the generation of behavior. This requirement implies that an implemented agent with a ToM should be able to explicitly represent the beliefs, goals and other mental concepts that it attributes to others.

In all of the described uses of agents with a ToM, the agents reason with others’ mental concepts, i.e. they predict actions on the basis of attributed goals and beliefs. For example, an agent that believes that agent B has belief X and goal Y, and that agents such as B with X and Y intend action a, should be able to derive that agent B probably intends action a. Based on its expectation of action a, the agent can select its own actions and thereby show the trainee how it interpreted his behavior (section 2.1), challenge the trainee to detect the flaws in its reasoning (section 2.2), or give the trainee support (section 2.3). It should be noted that a ‘normal’ reasoning process results in an actual action that is executed in the environment, but ToM-based reasoning should only result into an expected action which is not executed. In other words, the simulation of someone’s reasoning process should not have consequences for the environment. Thus, the implementation must allow agents to reason with attributed mental concepts, without affecting the environment.

To support the trainee, an agent uses its ToM to derive the cause of a trainee’s wrong or failing behavior, e.g. an incorrect belief or the lack of a goal. Therefore, the agent should not just reason with attributed beliefs and goals, but also with ones that are possibly attributable. For instance, the agent should be able to predict what agent B would do if it would believe X and have goal Y, without actually believing that B has that belief and goal. By comparing predicted actions to the trainee’s actual ones, the agent can diagnose the probable causes for his behavior and react on that. For the implementation this means that agents with a ToM must be able to reason with different combinations of not (yet) attributed goals and beliefs.

Finally, agents should use their ToM for the generation of behavior in order to add value to a training system. Only then, trainees can be given feedback, challenged or supported. Thus, the agent program should contain actions that are involved with updating, querying or reasoning with the agent’s ToM. To give support for instance, the agent program should include a rule that if the trainee is not acting for more than X seconds, the agent determines with its ToM what the trainee should do and how it can bring about the desired result.

4. IMPLEMENTATION OF A TOM

In this section we show how the requirements on agents with a ToM discussed in section 2.4 can be met by implementing them in a BDI-based agent programming language which allows for modularity. There are several BDI-based agent programming languages that allow for modularity, e.g. Jack [4], Jadex [3] and extended 2APL [9]. In these proposals, modularization is considered as a mechanism to structure an individual agent’s program in separate modules. Each module contains mental elements such as beliefs, goals and plans, and might be used to generate behavior in a specific situation.

Of these approaches, the approach in extended 2APL provides agent programmers with most control over how and when modules are used [9], and therefore we use extended 2APL to illustrate our approach of implementation agents with a ToM. Currently, the extension of 2APL by modules has not been fully realized yet, but we follow the definitions of extended 2APL as given in [9] (for an overview of ‘normal’ 2APL see [8]). Besides, we suggest some additions to the proposed modular approach of extended 2APL, to make it appropriate for developing agents with a ToM.

In this section we first introduce a modular approach to implement agents with a ToM and show its advantages over non-modular approaches. In the second subsection, we discuss how agents can develop their ToM. For instance, when an agent observes actions of other agents it should be able to update its ToMs about those agents. Subsequently, we discuss how an agent should use its ToM in order to determine its own behavior, i.e. its assumptions about other agents’ beliefs, goals or expected actions influence its own actions.

4.1 A module-based approach

A 2APL agent has a belief, a goal, and a plan base containing the agent’s beliefs, goals and plans, respectively. An agent’s beliefs, goals and plans are related to each other by a set of practical reasoning rules. A typical 2APL reasoning rule has the form: Head <- Guard | Body, in which the Head and Guard are tests on the agent’s goal and belief base, respectively. The Body of the rule contains one or more actions, which can be abstract, decomposable actions (plans), or atomic, directly executable actions. In 2APL, beliefs in an agent’s belief base are treated as Prolog facts and rules, such that the belief base of a 2APL agent becomes a Prolog program.

For the implementation of a ToM, we use extended 2APL, i.e. 2APL extended by modules, as introduced in [9], and a ToM of another agent is represented in a module. If no modules were used, an agent could only represent mental states attributed to other agents in its belief base. Only there, arguments can be added to the attributed mental states to distinguish them from the agent’s own beliefs, and to distinguish different attributed elements from each other, e.g. beliefs and goals. For instance, without modules, the beliefs belief(B,X) and goal(B,Y) could be used to represent that an agent believes that agent B believes X and has goal Y.

The disadvantage of implementing a ToM in an agent’s belief base, however, is that the interpreter of the agent program does not recognize attributed plans and goals, but only beliefs. Then, in order to reason with attributed mental concepts, epistemic reasoning rules have to be added to the agent’s belief base. A rule that makes combinations between beliefs about another agent’s beliefs and goals is for example: plan(B,P) :- belief(B,X), goal(B,Y). With this rule, the agent can derive that if it believes that agent B has belief X, goal Y and reasons according to Y <- X | P, agent B’s plan is P. However, this requires a translation from a supposed BDI program with practical reasoning rules to a Prolog program with epistemic reasoning rules. In contrast, when attributed mental states are represented in a module, attributed goals and plans can be represented and thus recognized as such, and the agent’s own deliberation power can be used to reason with attributed mental states. Besides that the module-based solution is more practical, it saves
work for the programmer.

In order to reason with the attributed mental concepts of an agent, the attributing agent can execute the module of that particular agent. However, the outcomes of the reasoning process should have no direct consequences for the agent’s environment, i.e. actions resulting from the reasoning should not be executed. The difference between executing an actual program and a module is that in the former derived actions are always executed and in the latter this is not necessarily the case. With a “dryrun” execution action, a module is executed without consequences for the environment and other agents. The state of a module after execution can be queried to estimate what the agent will do and what its active goals are. In extended 2APL for example, the command agentB.dryrun(ψ) executes agent B’s till stopping condition ψ is reached. The dryrun function has been proposed in extended 2APL, but not completely specified yet.

Each module can own its own modules, which allows for modeling agents that believe that other agents also have a ToM about other agents, which in turn might have a ToM, etc. The different levels of depth in reasoning about other agents are called orders of ToM. A first order ToM defines someone’s beliefs, thoughts and desires that influence one’s behavior, e.g. ’she does not know that her book is on the table’. In a second order ToM it is also recognized that to predict others’ behavior, the desires and beliefs that they have of one’s self and the predictions of oneself by others must be taken into account, e.g. ’she does not know that I know her book is on the table’. To have a third order ToM is to acknowledge that others have a second order ToM, etc. Though it is possible to implement higher order ToMs following our approach, they quickly become complex and impractical. However, as already mentioned, there is evidence that humans only use first-order ToM most of the time [19], and it is thus not necessary to have agents with a high order ToM for creating realistic human-like behavior.

### 4.2 Using a ToM

The beliefs and goals in a ToM can be present from the beginning of a scenario or obtained during interaction. In general, agents already have some knowledge about other agents before they interact. The context in which interaction takes place gives information, e.g. someone at the department hall of an airport probably has the goal to go to some place. In virtual training for incident and crisis management, the roles of the participants in the scenario give a lot of information about their probable mental states. For instance, a firefighter most probably has the goal to extinguish fires. The tasks connected to these roles in the intended domain are highly procedural in nature; for each possible situation is described how one should act. Such pre-known information in a ToM is expected to be stable over the course of interaction, e.g. during one training session, and thus does not need to be updated on line.

Agents attribute new mental states to other agents during interaction based on their observations of the environment. In a shared environment one can expect that the other has similar beliefs about the environment as oneself. For instance, the ringing of a telephone is most probably heard by everybody in the same room. In extended 2APL, a module of agent B is updated with a belief X by agentB.updateBB(X), and the module of agent C is updated with a goal Y by agentC.adoptgoal(Y). Information obtained during interaction can remain the same for the rest of the session, e.g. the belief ‘there has been a fire alarm’. And other attributed beliefs such as ‘I am at location X’ will most likely change within the course of the session. A trade-off has to be made between conservatively developing ToMs, i.e. only adding new beliefs and goals in case one can be quite sure that they are correct, or not. The former implies for the greater part correct ToMs with little information and the latter more extensive ToMs with the risk of being incorrect. Dependent on the context, the extend to which ToMs may be incorrect can be chosen.

Given that a ToM module contains some reasoning rules, goals, beliefs or plans, an agent can use a ToM by letting its decisions also depend on checks on its ToM. In other words, the results of a check on one or more of its modules are input for the agent’s own deliberation process. For instance, agent B only adopts goal G if it believes that agent A has a specific belief or goal, or is going to perform a particular action. In the remainder of this section we explain the general method by which an agent can use its ToM.

As discussed in section 4.1, a typical 2APL reasoning rule has the form: Head <- Guard | Body. Normally, to implement that a goal G has a subgoal G’ which adoption depends on conditions C, the following code is used [12].

\[
G \text{ and not } G' \text{ <- } C \text{ | adopt}(G')
\]

The rule ensures that if an agent has goal G, it can only adopt G’ if G’ has not already been adopted and conditions C are derivable from the agent’s belief base. The above rule can be used when condition C only contains possible beliefs about e.g. the agent’s environment. However, if the adoption of G’ also depends on what the agent believes that another agent believes, an extra step is needed. Namely, in order to obtain up-to-date beliefs about the other agent, an agent must check its ToM of that agent, possibly involving the updating or execution of the ToM module. Checking the ToM before determining whether G’ should be adopted can be accomplished by the following two rules.

\[
G \text{ and not } G' \text{ <- not checkedToM}(A,G') \text{ | checkedToM}(A,G') \\
G \text{ and not } G' \text{ <- checkedToM}(A,G') \text{ and } C \text{ | adopt}(G')
\]

These two rules ensure that if an agents has goal G, before it possibly can adopt subgoal G’, a check on the agent’s ToM is performed. Initially, the agent does not have a belief checkedToM(A,G’), meaning that the ToM of agent A concerning goal G’ has been checked, and only the first rule can apply. The body of the first rule contains a plan checkToM(A,G’) for checking the agent’s ToM about agent A, concerning goal G’. The execution of this plan adds two beliefs to the agent’s belief base: the result of the ToM check, and the belief checkedToM(A,G’). Thus, after the execution of the plan checkToM(A,G’) the first rule no longer applies, and the second rule may apply. Application of the second rule and thus the adoption of goal G’ depends on whether the conditions specified in C are derivable from the agent’s belief base, in this case involving beliefs about agent A’s mental state.

To actually check a ToM, the agent requires a plan for each goal of which the adoption of that goal depends on
the agent’s beliefs about another agent, i.e. a plan checkToM(A,G'). Such plans must per goal and agent specify in which way that agent’s ToM has to be checked. In general, a plan for checking a ToM involves the following steps: updating, executing and querying the ToM module, and finishing the ToM check, of which the first two are possible steps. An example of a plan for checking a ToM in extended 2APL is as follows:

```prolog
checkToM(A,G') <- true |
{ 
agentA.updateBB(b); 
agentA.dryrun(psi); 
if agentA.P(a) then 
UpdateBelief(agentA(a)); 
UpdateBelief(checkToM(A,G')) 
}
```

The first step of the plan checkToM(A,G') is to add belief b to the ToM module of agent A. In the next step, the module of agent A is executed till stopping condition psi is reached, which is e.g. that the first external action has been determined. Subsequently, by agentA,P(a) the plan base in the ToM module of agent A is queried about whether it contains a plan a, and if this is the case, the belief agentA(a) is added to the agent’s belief base. Finally, the belief checkToM(A,G') is added to the agent’s belief base. In the agent’s main program, one of the conditions that must be true to adopt goal G’ is agentA(a).

All plans for checking a ToM involve a step in which the ToM module is queried, and all end by the addition of a belief denoting that the ToM of a particular agent for a particular goal has been checked. However, there are different ways to check a ToM, delivering different kinds of information. In the next section we discuss five ways in which a ToM can be checked.

### 4.3 Different ways to check a ToM

The *first* way in which an agent can check its ToM is to query the attributee agent’s believed belief base, possibly after updating and executing it. For example, in extended 2APL the action agentD.B(X) queries whether belief X can be derived from agent D’s believed belief base. The result should be added to the agent’s own belief base, and dependent on the result the agent can adopts a goal or not.

The *second* possible check is to query the attributee’s believed goal base, possible after updating it. In extended 2APL, agentD.G(Y) queries whether it is believed that agent D has goal Y. Here again, the result of the query should be added to the agent’s belief base.

*Third*, a ToM can be used to predict what another agent is going to do. To answer that question, the ToM module of that agent has to be executed, after possible updates, till the first action that the agents is expected to perform is determined. As mentioned before, the execution is a dryrun execution so that the resulting action will not actually be performed. Subsequently, the plan base in the module can be queried, and beliefs about plans and upcoming actions can be added to the agent’s own belief base. The code in the previous section with a plan for checking a ToM is an example of this third possible way of checking a ToM.

Note that is only possible to execute a module till the first external action is determined, because execution of the action would change the environment. Feedback from the environment about its new state is necessary to determine the next action, however, simulating someone’s reasoning process should not have an effect on the environment. Currently, its is not possible to query the plan base of a module in extended 2APL, this capability should be added to the language.

The *fourth* use of a ToM involves the interpretation of someone’s actions, that is, observed agent actions are explained by a ToM. For example, when an agent suddenly stops extinguishing a fire, it might have seen a victim or something that causes explosion danger. In order to use a ToM according to this fourth way an agent needs: one or more observed actions, a ToM with already some information about the other’s mental state, and a set of possible explanations.

Depending on the situation, agents have more or less knowledge on specific elements of another’s mental state. For example, in a game like chess, the opponent’s main goal and its beliefs are completely clear, that is, winning the game and the positions of the pieces on the board. However, the plans with which it wants to achieve this goal, and the reasoning rules it uses are harder to estimate. In contrast, for the guesser in a game like mastermind, the other’s beliefs are exactly what it does not know but needs to find out, i.e. the colors and positions of the four pieces. With information about part of someone’s reasoning elements and observations of his actions, missing information can be derived and added to a ToM. Without any information it is impossible to interpret someone’s actions and extend the ToM.

Besides a ToM with a number of attributed mental concepts, an agent needs to have knowledge about possible explanations for someone’s actions. In each context, particular events are likely to take place and of influence on agents’ actions; these events are the possible explanations for someone’s behavior. Following from the previous paragraph, in some domains possible explanations consist of beliefs (e.g. a mastermind game), and in others they consist of plans (e.g. a chess game). Per application, a set of possible explanations has to be defined, i.e. a set of beliefs or goals or plans. In the domain of crisis management, agents usually have information about others’ possible goals and plans because these are connected to their roles. The agents’ beliefs which determine their goals and plans, in contrast, are expected to be constantly changing during an incident scenario: an incidents develops, the agents obtain more and more information about the incident, and the agents are actively trying to change the situation.

Once an agent has the disposal of one or more observed actions, a ToM (a module) and a set of possible explanations (set of beliefs or goals), it can interpret an attributee’s actions. To do so, the agent has to ‘try out’ different possible explanations by, for each possible explanation, temporarily adding or removing a belief or goal to the ToM module and perform a dryrun execution with the new module. The actions that are outcomes of the dryrun executions are compared to the attributee’s actual actions. If there is a match, the particular alternation to the ToM probably explains the attributee’s actions, and the temporary change can be incorporated in the ToM permanently. The best explanation of someone’s actions can be added to the agent’s own belief base, and used to determine its own behavior. For this capability, extended 2APL has to be extended by the possibility to temporarily update a module, execute and query it, and
then make the temporal update undone.

The fifth use, like the fourth, involves the addition or removal of different combinations of beliefs or goals to a ToM. The resulting actions after executing the modules are not compared to actual actions of the agent, but to desired actions. This is useful when the agent wants to make an agent act in a certain way, i.e., socially manipulate the agent. By trying different combinations of mental states, one can find out what an agent must believe or which goal it must have to display the desired behavior. This information can help the agent bring about this behavior. For the last two uses of a ToM, an agent’s ToM must be updated temporarily with certain beliefs or goals, but after execution and one or a few queries, the updates must be made undone.

5. RELATED WORK

Bosse et al have introduced a formal BDI-based agent model for Theory of Mind [1]. In our approach, the attributing agent is BDI-based, and its ToM about other agents is also in terms of beliefs, desires and intentions. Bosse et al especially focus on how the agent can use its ToM in its own BDI-based reasoning processes with the purpose of social manipulation. In this paper, we have focused more on the implementation of a ToM and only shown the possibilities of using it, which also involves social manipulation. A difference is that in their approach agents only reason about attributed mental concepts, and in our approach agents reason with attributed mental concepts as it were their own. Thus, as argued in section 4.1, in our approach the deliberation power for normal reasoning is also used for reasoning with attributed mental concepts.

PsychSim is a simulation tool for modeling interactions and influence involving agents with a ToM [22]. PsychSim agents have a decision-theoretic world model, including beliefs about their environment and recursive models of other agents. Instead of using qualitative BDI concepts, they use quantitative models of uncertainty and preferences. According to them, a quantitative model is required to resolve the ambiguity among equally possible, but unequally plausible or preferred options. We take a BDI approach because it gives insight into the agents reasoning processes, which is desired in training applications. In our model, information about preferences is included in the agents’ BDI models, and between equally preferred options is chosen randomly.

In section 2.4 we have discussed several ways to develop a ToM, among which the ascription of mental states to an agent based on its actions. This phenomenon is also called intention or plan recognition and has been investigated by several researchers. Models of plan inference all start with a set of goals that an agent might be expected to pursue in the domain and an observed action by the agent. Most plan recognition systems infer an agent’s goal from the observed action by the agent. The difference between these systems and our approach is that they start with the observed action and infer possible sub-goals and goals, and we start reasoning with possible goals and determine which actions could result from them. The disadvantage of our approach is that it requires more steps because all possible goals have to be considered, whereas most other systems can exclude possible goals. However, the advantage of our approach over others is that it makes use of an agent’s existing reasoning strategies, and no extra plan inference system has to be built.

One of the uses of agents with a ToM is to provide sophisticated explanations about behavior. There are several accounts of self-explaining agents in virtual training systems, e.g., Debrief [16], XAI version I [26] and II [7], and our own [12]. In these systems, trainees get the opportunity to query agents about their behavior in the played training session. The explanations aim to improve trainees’ insight in the training situation. The kind of questions that the agents are able to answer vary from systems to system. Debrief reveals what must have been the underlying beliefs of an agent, XAI I informs about the state of an agent at a particular time, e.g., its position or ammunition, XAI II answers questions about agents underlying motivations that are made available by the simulation, and our methodology delivers agents that give explanations in terms of their underlying beliefs and goals. None of the current accounts of self-explaining agents provides explanations involving attributed mental states.

6. CONCLUSION

In the present paper we have examined the possible benefits and uses of agents with a ToM in virtual training. We have given different examples of how agents with a ToM can be useful in such applications, and illustrated that training with ToM-based agents is valuable in particular to train on correctly interpreting others’ behavior and becoming aware of how one’s behavior is interpreted by others. We have introduced a modular approach for the implementation of ToM-based agents. The proposed approach to implement a ToM allows an agent to reason with attributed mental concepts instead of about them which implies that an agent can only attribute mental concepts to other agents that it can have itself. As a result, ToM-based agents can fully make use of their deliberation power when reasoning about other agents.

In section 2 we have introduced the two most prominent current accounts on how ToMs work in humans: theory-theory and simulation theory. In our approach, after checking their ToMs, agents do obtain beliefs about mental states of other agents in their own belief base, which corresponds to the vision adhered to by theory-theorists. However, reasoning about nested beliefs and goals is done by simulation, which has similarities with the simulation theoretical view. In conclusion, our approach involves aspects of both theory-theory and simulation theory. In our opinion, an agent needs to have some preprogrammed constructs in order to be able to develop a ToM. However, it may use its own reasoning power to reason with mental concepts attributed to another agent, i.e., simulate the reasoning process of another agent.

We have illustrated the implementation approach with examples in extended 2APL, a version of 2APL extended with modularity as proposed in [9]. The examination of uses of agents in virtual training systems delivered a number of new requirements to extended 2APL which were not foreseen in the original proposal. Namely, to implement agents with a ToM a dryrun function, the possibility to query a module’s plan base, and the possibility to make temporal updates to a module are needed. We are currently working on the development of 2APL extended with modules, including these extra functionalities. After extended 2APL has been developed, we will implement actual agents with a ToM and apply them in a virtual training system.
able to conduct user experiments, which should show the use of agents with a ToM in virtual training systems.

7. REFERENCES


A Framework for Teaching Multiagent Systems

K.Rajendran
University of Teesside
Middlesbrough
U.K. TS1 3BA
+44 (0) 1642 218121
k.rajendran@tees.ac.uk

S.C.Lynch
University of Teesside
Middlesbrough
U.K. TS1 3BA
+44 (0) 1642 218121
s.c.lynch@tees.ac.uk

ABSTRACT

In this paper we acknowledge the difficulties in teaching practical development of multiagent systems and also those complex application areas, such as multimodal systems, which are typically supported by agent technologies. These areas are challenging due to the high learning curves associated with them and the extent of pre-requisite knowledge required for both learners and educators.

Our objective is to reduce these difficulties, targeting teaching towards undergraduate and postgraduate computing students. To achieve this we specify an adaptable multiagent architecture and describe how its internal design and supporting tools reduce the cognitive burden on learners. This architecture allows us to provide a set of general purpose, extensible agents to handle essential multimodal tasks like speech I/O, fusion and semantic analysis. We outline the design of these agents and describe how they provide a framework for students to assemble complex systems and experiment with agent-level design patterns and with MAS assembly in general.

We evaluate the usability of the resulting software and tools using the Cognitive Dimensions Framework and by examining students’ experience of using this approach in computing courses and project work.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education – Computer Science Education;
I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence – Multiagent systems, languages and structures;
D.3.3 [Programming Languages]: Language Constructs and Features – Frameworks;

General Terms
Design, Languages, Experimentation, Human Factors.

Keywords
Multiagent systems, multimodal dialog, design patterns, teaching.

1 INTRODUCTION

Multiagent systems (MAS) provide a level of abstraction above that of object orientation [5, 10, 20]. The research community has produced a choice of agent languages, platforms and frameworks to support MAS but these have been developed to support particular notions of agency and to address the requirements of developers rather than to meet the needs of students. Adopting an existing agent language for teaching has the benefit that students learn to use a system which has a wider user-base and acquire knowledge which may be useful to them in future but this must be offset against any limitations imposed by the system and/or issues of learnability. Many languages (eg: Jack, Jason, 3APL) embody specific flavours of agency (BDI for example) which constrain the types of agents that they can specify and limit the possibilities for students to experiment with agent concepts in general [1]. Additionally, many existing agent languages present difficulties for learners at both the syntactic and conceptual levels because their design is based on theories of agent behaviour rather than on pre-existing knowledge we would expect from students. In some cases this situation is made more difficult because of the lack of tools available to help develop and debug agent systems.

To avoid these problems we have developed a JAVA-based agent language designed specifically for learnability. In addition to our aims of providing an agent language that novices can use to build their own agents, we are also interested in providing a framework suitable for students to assemble complex/challenging systems from pre-existing agents. Due to our other research and educational interests we select multimodal dialog (MMD) systems as an application area and provide a small library of predefined agents for this. Modern complex applications such as multimodal systems are typically developed jointly by experts in a variety of technologies. Even in research environments, the development of prototype systems presents a “demanding challenge” [9]. Incorporating practical MM methods and technologies into courses can be difficult for several reasons. Firstly there is a requirement for varied and specialist technologies such as speech recognition and gaze tracking, using these is difficult without specific expertise. Secondly the processes involved in using a combination of modes to capture inputs e.g. speech and facial expression, in parallel can be complex. These are processes like multimodal fusion and fission which are specialist research areas in their own right. Also these systems are often dialog-oriented or “conversational” which makes the processing of inputs considerably different to traditional systems that process unconnected discrete commands since they require processes like conversation tracking and maintenance of context. Finally there is an inherent difficulty in building systems which are complex, distributed and involve heterogeneous components.

Previous research into multimodality has resulted in a number of generic architectures and components to simplify development
e.g. [3, 7, 9, 11], many of these architectures take a MAS approach. Researchers have shown the advantages of applying MAS techniques to complex systems development in general [10] as well as MMD systems in particular [15] but although they have shown the possibility of re-use they are almost entirely in the research domain and suffer from poor usability and offer little real practical re-use. This means that multimodal systems are still typically beyond the scope of project work for undergraduate students and even postgraduate students following taught courses.

Much research into multimodality is focused on education [14, 16, 17] but these efforts aim to improve teaching and learning in general by using multimodal systems rather than advance the teaching of multimodal systems construction. Conversely, our research aims at simplifying practical development of multiagent systems to a level suitable for courses at university and subsequently to use MAS technology to simplify teaching and learning of multimodal dialog systems.

In this paper we present a MAS platform and API, supporting tools and a set of agents for multimodal systems. The API does not enforce a particular view of agency and offers a range of possibilities for experimenting with agent collaboration and communication. We have designed this platform for ease of use and learnability and identify some specific learning objectives:

(i) students are able to program their own agents and build simple MAS applications;
(ii) students are able to assemble/configure MMD systems by using and tailoring pre-existing agents and considering related concepts like design patterns;
(iii) students are able to experiment with a variety of agent concepts (including types of messaging, synchronisation, etc).

The next section introduces the agent-ware that supports the teaching approach described here. Section 3 describes a suite of agents to handle some of the key tasks involved with building multimodal interfaces. Section 4 identifies a skeletal but extensible agent architecture for MMD systems and also smaller architectural patterns. Sections 5-6 conclude with evaluation of the approach.

2 MAS FRAMEWORK

We present a MAS framework that has been designed for usability and learnability with the aim to reduce the complexity of practical development to a level suitable for final year undergraduate or post graduate courses. The associated MAS platform is generic enough for a variety of application areas and it is not specific to any branches of agency. It allows simple, as well as more complex applications through easy extension mechanisms and interchangeable modules. There are specified models of extension for different application areas as well as for adding support for different programming languages. The toolkit associated with the platform provides several visual tools to aid learners which may be extended to provide increased support for new application areas. This means that MMD specific tools can be built as extensions of the MAS toolkit.

2.1 Programming Agents

The platform presented here is used with an agent API for programming agents that will function as part of the MAS. The MAS application’s components and the MAS structure are created using a visual toolkit. The agents are then ready to run.

We have implemented stubs of our agent framework in Java, Lisp and C#. For each implementation we have used the Cognitive Dimensions Framework to guide choices for the syntactic style used to specify agents and have additionally made effort to provide a simple but coherent analogy between the way agents are represented in any language and other language features. Inevitably this leads to different syntactic structures in different languages – the specification of Lisp agents, for example, is quite different to that used for java agents but each fit within the style of their native language. Also, the API is easy to understand because it matches the generic agent concepts like agent and message.

To ease the burden for new learners, our multiagent framework presents agents as software components which are triggered when they receive messages. In this sense we consider agents responding to message events and maintain a close analogy between agents and other objects which respond to events.

Consider, for example, a GUI Button – this can be specified in Java code as follows:

```java
Panel p = new Panel();
Button b = new Button( text );
b.addActionListener( new ActionListener()
{   public void actionPerformed(ActionEvent event) 
{   ...code body...
} 
});
p.add( b );
```

The code above declares a panel and a button, it associates an event listener with the button (specifying the event handler method which is called actionPerformed for Buttons' ActionEvents) then adds the button to a GUI container (a Panel in this example), this allows the button to be visible to users.

In our framework there is an analogy between agents and GUI components (Buttons, etc) and between GUI containers, which make the components visible, and Portals. Portals provide communication channels for agents making them visible to other agents and the MAS as a whole. This analogy not only holds conceptually but also in the appearance of program code necessary for specifying agents.

```java
Portal p = new Portal( portal-name );
Agent a = new Agent( agent-name );
a.addMessageListener( new MessageListener()
{   public void messageReceived(String from, String to, String msg, MsgId id) 
{   ...code body...
} 
});
p.addAgent( a );
```

The code above declares a portal and builds a new agent. It associates a MessageListener with the agent, specifying the messageReceived handler method for it. The MessageListener is an event listener which will trigger when the agent receives a message and will activate the messageReceived method to deal with the new message.

As shown in the code above, agents respond to message events, with a handler method called messageReceived which takes the following arguments:
from the name of the agent which sent the message;
to the name of the agent that the message was addressed to.
This is normally only useful if the same Message-
Listener is shared between agents;
msg the text of the message;
msgId a unique (and faceted) identifier for the message, which
 can be used for tracking conversations and session
management.

In addition to having behaviour associated with receiving
messages, agents are also capable of sending messages, the
simplest mechanism for this, in Java, is to use sendMessage which
has the following specification...

```java
void sendMessage( String to, String msg )
```

To send the message "have a nice day" to an agent named "sid",
for example, the call would be...

```java
sendMessage( "sid", "have a nice day" )
```

We aim to provide a platform which is simple enough for use in
second year programming courses but also provides those
facilities necessary to underpin more advanced work in MMD
and aspects of Intelligent Systems. In these, more complex systems,
messages are often sent in a series of exchanges which together
form some kind of dialog, these exchanges are known as sessions.
A typical session may be initiated with sendMessage but then
continued by sending replies to earlier messages (or earlier replies).
To facilitate this type of behaviour we provide a
mechanism to send replies to messages and also to listen for
replies. Replies are sent (in Java) using...

```java
sendReply( MsgId replyTo, String reply )
```

The first argument for sendReply is the MsgId of the message that
is being replied to. This is most often a message that has been
recently received. The underlying framework manages the task of
ensuring that the message gets delivered to the correct agent.

The code below defines a simple agent class which send a reply
"thanks for the message" for any message it receives...

```java
class MyAgent extends Agent
{ public MyAgent( String name )
{ super( name );
}
public void messageReceived( String from,
String to, String msg, MsgId msgId )
{ sendReply( msgId, "thanks for the message" );
}
}
```

Replies are handled by ReplyListeners in a similar way that
messages are handled by a MessageListeners, using a different
type of listener for replies introduces a new event handler but
makes it easier for agents to track multiple sessions.
ReplyListeners use a handler method call replyReceived with the
following signature...

```java
replyReceived( String from, MsgId mfor, String
reply, MsgId id )
```

The replyReceived method takes 4 arguments, these are...

from the name of the agent which sent the message;
mfor the MsgId of the message that this is a reply to;
reply the text of the reply;
msgId a unique (and faceted) identifier for the reply, MsgIds
are described later in this document.

Replies are only sent to those agents which request them. An
agent requests a reply by including a ReplyListener as a third
argument in a call to sendMessage or sendReply. The
ReplyListener and its associated replyReceived method is
automatically invoked when a reply is received, eg:

```java
ReplyListener rl = new ReplyListener()
{ public void replyReceived( String from,
MsgId mfor, String reply, MsgId id )
{ // place code to handle the reply here
}
};
sendMessage( "sid", "have a nice day", rl );
```

In the above example the replyReceived method will be called
when "Sid" sends a reply to the "have a nice day" message.

Message id.s are faceted (ie: they have different parts), most of
these are only ever used by the agent kernel but one facet is the
session id. The session id is a unique string identifier which can
be used to identify any given session. All messages which are part
of the same session will have the same session id and every
session will have a different id.

Programmers writing agents to handle sessions in some
specialised way may need to access session id.s, they can be
retrieved with MsgIg.getSessionId(). Together with the use of
sendReply and replyListeners this facilitates various ways to deal
with and experiment with agent-agent dialog.

### 2.1.1 Sending messages /clones & non-clones

From an educational perspective we wish to present a variety of
possible agent architectures and interactions. For this reason we
try not to constrain the types of agents that can be developed or
the nature of agency – we would not, for example, want to
constrain agent development by using only BDI agents. In order
to maintain a flexible approach we allow agents to be cloneable or
non-cloneable and also permit some styles of message handling
which are absent from many MAS frameworks.

Cloneable agents process their messages as soon as they receive
them. If a cloneable agent is busy with an existing task a new
clone is created to process the new message. Non-cloneable
agents have their messages queued when they are busy.

Message replies for both cloneable and non-cloneable agents are
routed through ReplyListeners but the scheduling of this activity
differs. With cloneable agents, ReplyListeners run concurrently
with other ReplyListeners and also with any other activity of any
given agent. With non-cloneable agents, ReplyListener activity
occurs non-concurrently, in sequence with other ReplyListeners
and also with any other activity of any given agent. This
sequential activity is scheduled on a first-come first-served basis.

In practice clones are handled by creating a new process thread
each time their MessageListeners are triggered. Clones share
global variables (including instance variables for agents which
extend the Agent class) but local method variables are not shared.
This type of behaviour only allows multiple clones/duplicates of
an agent to exist simultaneously. Some authors refer to
cloning/duplication to imply a complex process with a strategic
underpinning [4] here we use the term clone/non-clone only to imply a simple modification to behaviour – the term clone does not imply fault tolerance or any replication of data (instance variables, etc).

The provision of clones allows students to build different types of service agents as well as agents associated with a variety of other software components. It introduces variation to the implementation of agency and allows students to investigate new aspects of agent-agent synchronisation.

In some patterns of agent-agent collaboration it is possible that one agent may delay further activity until it has received some kind response from another agent (an example of this could be if an agent is waiting for contact details from a broker for example), but in normal models of agency the waiting agent would not be suspended at the process/thread level. Its process would stay active but it would not continue with its current tasks until a reply had been received. There are good reasons for this model of operation (agents are autonomous/independent entities, deadlock is avoided, etc). However, in gaining the ability to provide a platform for students to build well engineered systems, we also want to encourage them to experiment with aspects of concurrency and agent synchronisation. For this reason we also provide alternative forms for sendMessage and sendReply which suspend the current agent's process/thread until a response it received.

2.2 Visual Toolkit

Ethnographic evaluations which involved observations and interviews with students have shown that the main difficulties students face with both writing individual agents as well as assembling MAS is the run-time debugging of MAS activity. We initially provided a message capture and display tool which evolved alongside iterative evaluations to become a toolkit/IDE which acts as a visual front end to the platform (Fig.1). The following sections outline its main functions.

2.2.2 Messages Display

Interactions are a key aspect of the MAS and debugging MAS includes debugging interactions. Observing interactions also allows students to confirm that their system works as expected. Functions involving filtering like conversation tracking [6], verification of scenarios by matching messages against design and the ability to restrict logs to messages relating to particular agents allow students to focus on certain aspects as well as providing a mechanism for examination of student work.

2.2.3 Logging and Playback

It is difficult to observe MAS in real-time because system activity occurs too fast for human comprehension. Traditional programming tools have used features like breakpoints to pause execution so that current system states can be inspected but these facilities are impossible to use with distributed, concurrent systems. The toolkit capture MAS activity (e.g.: agents joining the MAS, messages and errors) so that it can be replayed slower speeds which allows breakpoints to be applied on the replay mechanism. These kinds of activity is currently not well supported although their importance has been highlighted for some time.

2.2.4 Test Harness

The toolkit allows agent interactions to be driven manually (or through scripts) to test either single agents or sets of agents without the presence of other agents that send or receive messages from them. This is needed because related agents may not exist or have been developed. Some systems provide messaging agents for this (e.g.: JADE's "Dummy Agent", Mock Agents in Agile PASSI).

2.2.5 Launching

Observations showed that support for starting agents as well as groups of agents and MAS encourages students to experiment more because starting up agents individually or through code can be error-prone. MAS consist of independently executing entities and do not have a single starting point so launching them is complex and error prone. Information to support launching include details such as agent instances (type and number of agents), locations of agent executable files, their dependencies (constraints on the order in which they are started up) and structure (hierarchical groupings). The toolkit allows these to be specified as several variations and started up through a menu.

2.2.6 Link to Existing Tools

Since many code production tools like editors already exist and are suitable for writing agent code, the MAS toolkit does not provide them but instead allows links to existing tools. Studies have shown that there is a lower perceived learning curve when students are allowed to use tools (like editors) they are already familiar with.

3 AGENT LIBRARY

3.1 Speech Agents

Speech recognition technology is rapidly making significant progress but several challenges still exist [2]. Some of these relate to speed and accuracy, others to a lack of standards and suitable
architectures that allow reuse across applications. Our work has focused on the latter, aiming to provide reusable components in the form of agents. We chose speech input/output which is a popular and highly supported modality as the focus of the study. Speech recognition also introduces the need to develop language artefacts like grammars and lexica which could not be omitted from any MMD architecture claiming to be of general use.

As mentioned before, complex systems like multimodal systems require the contribution of various subject experts and it is not practical for individual students to develop enough knowledge to build these systems from scratch. We identify two types of learning activity, in one scenario, students learn to build particular elements, in another, they learn about assembling systems from re-existing elements. The objectives are to provide support for both types of learning activity in various ways:

(i) exploring assembly of complex systems and related concepts like design patterns;

(ii) as a supporting framework for students studying certain aspects in depth. In this case their learning experience is improved by being able to “plug-in” in their work to a quickly assembled scenario which puts their work in context allowing them to gain a better understanding of the wider application;

(iii) the learning associated with difficult aspects may be eased by providing agents that encapsulate layers of complexity allowing students to progress through different levels of abstraction and behaviour.

At the most simple level of use, one type of agent has the capability to communicate with the speech synthesiser and forward to it the text of all messages it receives. Once this agent is connected to the MAS, any other agents in the MAS may send it text to be spoken. Students are able to use this agent easily because it is self contained and executable. Also, as mentioned before the visual toolkit supports launching which means that speech output can be added to a MAS application without any programming.

At a more involved level students can also include speech agents into their code as shown below:

```java
SpeechOutAgent a = new SpeechOutAgent("speaker");
```

This creates an agent named “speaker” with all of the functionality of a generic agent. This agent also functions as a generic speech agent and is capable of communicating with the speech synthesiser installed on the physical machine. The code below shows how the agent may be used to produce speech:

```java
a.speak( "hello world!" );
```

This agent provides a number of functions that can be called for tailoring the speech to set the pitch of the speech, rate of speech, volume etc. In addition to the minimal speech input agents described above there is a suite of other speech agents. Some of these offer synchronisation with animation clips and/or other graphics. These can be further extended to develop complex behaviours like lip-synch.

Similarly, agents are also provided for speech input that sets up the recogniser, grammar and processing input in various ways. In speech enabled conversational systems, grammars are used at two different levels. They are used by the language understanding and generation software as well as dialog managers. These grammars may have semantic as well as lexical information. These tasks are difficult in their own right and increases the level of knowledge students must posses before building applications involving speech. Along with the speech input agents we provide utilities to synchronise the grammars used by speech and language agents (see later section).

### 3.2 Fusion

Fusion is an intricate process which needs to consider temporal issues as well constraints presented by the use of different modalities. Our observations of students show that, though they can outline ideas about the nature of a fusion process, they require significant support in order to produce a coherent design. We find that fusion (along with semantic analysis and, when appropriate, dialog management) is one of the main barriers for students' understanding of practical MMD.

The fusing process is necessarily highly dependent on the nature of input modalities used and format of data supplied by agents connected to these sources of input. This implies that a fuser will be closely coupled with input modes and, in many cases, to the application domain. Our aim, however, is to provide a generic fuser which will provide much of the necessary functionality (at least for small to medium scale systems) regardless of the input modes chosen and independent of the application domain. This presents a dilemma.

Broadly there are two approaches to fusion: early and late. Early fusion takes syntactic tokens from each of the input modalities and combines them to form new syntactic tokens. Late fusion assumes that input from each modality has been semantically analysed and combines semantic fragments to produce larger scale (more complete) semantics. In practice many researchers discuss hybrids between early fusion and late fusion, which either take input that has had some semantic processing on some input modes or may repeat fusing at different stages in the process of semantic analysis.

Specifying a generic fuser which will apply late fusion presents problems. Semantics are typically (and often necessarily) application dependent. Conversely, fusing based solely on low-level syntactic tokens is more easily achieved but these tokens are often typically highly related to specific input modes.

After various trials, we have developed a fuser in the following form. The core fusing process is independent of any application domain and can achieve early fusing and/or hybrid fusing (where input sources may have been semantically processed). It operates using a set of fusion rules which drive the fusing process. The advantages of separating information about how to fuse inputs from the actual task of fusing them have been recognised by other authors e.g. [19]. In the model presented here, the rule-set which is application dependant is separate from the fusing agent which means that students working on practical development projects are free to code application specific concepts into their rules without the need to reprogram the fuser itself.

Fusion rules specify how input from different modalities can be fused and the temporal and/or modal constraints which should apply. In a manner similar to the dialog manager described by Portillo et. al. [18], the fuser maintains a multimodal input pool of
input data. Input from different sources that overlap temporally (or nearly overlap) is considered as potentially multimodal and the fuser will attempt to combine the input according to its fusion rules. Fused input is then sent to a language processor.

### 3.2.1 Fusing Rules

The fuser typically receives information about spoken input and input from other modalities (gesture, etc). Each input of data to the fuser represents one utterance by the user. An utterance relates to one command or sentence the user says. These are identified by the speech input agents through reference to a defined grammar.

The input consists of the words that were spoken and their start and end times as recorded by the speech agents. The gesture input contains details of the objects that were referenced by the user (by pointing, clicking etc.) and the times associated with each reference. With pen-based input for example, this time would be the time the user pointed at an object with the pen.

There are various different ways that the fusion process can be developed but the goals are the same: to combine compatible, time-coincident data from different input sources when that data refers to the same semantic element.

As well as a need for the fusing process to be transparent and understandable (for the benefit of students) there is also a requirement that any resulting solution is reusable across different applications – we are interested in building a generic agent which can be reconfigured as necessary. We achieve this reusability by providing a basic inference mechanism which is guided by fusion rules that are declarative in nature and aim to be reasonably easy to write.

Consider the following case... the user says "that" and clicks over an object at the same time. Among other tuples/facts describing the multi-input utterance will be something like...

```lisp
(speech-in (that 1 5 20))
(gui-in g0 type $obj))
(gui-in g0 time 12)
(gui-in g0 target blue-block)
```

The first tuple is from a speech input agent named speech-in and relate to the user speaking "that". The numbers 2, 15, 25 have the following meaning...

1. the edge number, in this case that "there" was the 2nd word spoken (edge numbers start from zero);
2. the start time – the time when the recogniser started to receive the phonemes that make up the word (time data has been simplified for the sake of discussion);
3. the end time – the time the last phoneme was completed.

So the spoken word "that" is smeared over times 5-20 with a GUI click arriving at time 12.

A fusion rule could describe the following: when the word "that" is time-coincident with a mouse click and that click is over a GUI feature of type $obj rewrite the tuples describing this event as a new fused tuple describing the event. The specification of a fusion rule for this transformation would be as follows...

```lisp
(rule that-click
  (fuse (speech-in that) (gui-in $obj))
  => (fused (gui-in type) (gui-in target))))
```

This derives the single tuple below from those shown above...

```lisp
(fused ($obj blue-block 1 5 20))
```

### 3.2.2 Input Formats

Input agents (like those for speech input and GUI input) simply accept data and send it to the fuser. As described above, the fuser processes sets of tuples and assumes that it will receive data in this format. However this is not the same format that data will always be most obviously gathered by the input agents. To modify the structure of data we provide reformattting utilities which logically sit between the input agent and the fuser but are, in practice, provided with the fuser as part of its cluster of agents.

### 3.3 Semantic Analysis

The language processing agent provided is based on Lkit, a Natural Language Processing toolkit (available at www.agent-domain.org). Lkit includes a parser for carrying out syntax analysis and a rule based engine for semantic processing. All language-specific and application dependent structures for syntax analysis are coded into a lexicon and set of grammar rules. The semantic processing rules are also provided as part of the grammar.

The agent provided as the default language processor is a version of Lkit which has been wrapped in the form of a Lisp agent and will accept data in the form produced by the fuser. The other specialised feature involving Lkit is a utility agent which allows grammars and lexica to be shared with a speech recogniser.

### 3.4 Grammar Synchronisation

The language processor uses a grammar and lexicon to deconstruct a sentence and ultimately to build some form of semantic representation which captures its meaning. The speech recogniser uses a grammar and lexicon to improve its recognition by placing constraints on the sequence of words which can legally occur. This helps the recogniser select valid labellings for the phonemes which it decodes.

Unfortunately the grammar/lexicon used by language processors is not compatible with that used by recognisers. In addition to allowing grammar builders to specify semantics, language processors typically provide a richer notation for expressing grammar rules and support a greater level of complexity for rule forms. This means that students must become familiar with two different forms of language specification. For researchers who will spend considerable time modifying their language specifications this may not cause significant problems but for students wishing to experiment with MMD concepts as only a part of their wider programme of study it introduces the burden of learning additional syntactic constructions and also increased possibilities for errors – we have observed students using incorrect grammar formats or mixing formats.

To overcome these difficulties we supply an agent which will take a grammar and lexicon specified for the language processor and automatically produce a grammar file for a recogniser.

### 4 ARCHITECTURES

Example architectures are important reference tools for learners, especially for systems with complex internal structures. In a multiagent system, the MMD components are in the form of agents and the structure of the system, defined by the agent groupings and hierarchies as well as interactions between agents.
and/or agent groups, are of primary importance. Patterns can be seen in the arrangement of agents, interaction scenarios and higher level collaborations which have different advantages in different scenarios. Architectures may be presented at different levels of abstraction from a high level view of the system to expanded views which show more detail. This is a necessary property since students are often required to make in-depth study of specific areas but still have a general understanding of a whole system. As mentioned before, most learners will not gain in depth understanding of all system components but an understanding of abstract views of systems is possible. The contributions that can be made by reusable abstract patterns have also been suggested for other application areas of multiagent systems [12, 13].

There are various criteria to be considered when specifying example architectures:

i. they should be based on abstract and widely known concepts rather than specific features;
ii. they should be simple enough for learners to comprehend;
iii. they must be flexible enough to extend to real applications that are complex and large.

We present the diagram below (Fig.2) as an example high level generic MAS architecture for MMD. The diagram shows speech input and GUI input but is extendible to include additional and/or alternative input modalities.

Figure 2. Example architecture for an MMD system.

Figure 3 and 4 show "sub-architectures" that can be used as design patterns for building the different parts of the system. Each of these are independent architectures in their own right with their own structure and interactions and so they may be considered in isolation and developed as separable agent clusters which can then be linked together to create an entire application.

Students are faced with a smaller learning curve if provided with design patterns and a range of pre-existing agents (like those described next) which fill the roles of components in these patterns.

5 EVALUATION

We evaluated the usability of our MAS platform using the Cognitive Dimensions Framework (CDF) [8]. CDF assesses the usability of an artefact by considering a variety of factors including those related to learnability and the likelihood for new users to make mistakes. In particular we used CDF (i) to appraise the concept of agents and portals and the use of event handling as a suitable metaphor for conceptualizing agent message handling (ii) to design the syntactic specification of agents and their listeners. A result of this process, the code defining agents was designed to closely follow code defining GUI components.

In addition we monitored students' experience of building agents and MAS at the coding level (ie: without access to a library of pre-built agents). Students attempted various tasks including building chat-rooms, multi-user document editing facilities and a simulation of an agent based smart home. Students adapted quickly to the notion of agents as message-event handlers and began using agents to implement behaviour in situations where their use was not mandatory (ie: for other programming exercises).

As discussed earlier we selected multimodal systems as a vehicle for monitoring student performance with assembling MAS from pre-constructed agents. We chose multimodal systems because they offer a suitable test case and also because we are interested in running modules in this area. Our examination of results in this area focused on the usability of the underlying architecture and the multimodal components from the perspective of a novice developer. In particular we are interested in whether the approach we had taken allowed students to successfully experiment with practical deployment of multimodal systems.

Evaluation was conducted in two phases with small user groups of students who had some programming experience (in Java) but were unfamiliar with the concepts of multiagent systems, speech technology or MMD. In phase-I, groups of students were given tasks which involved developing applications using speech I/O agents. The groups were given an introduction to the general concepts of MAS and to the API of agents provided (including an outline of the speech agents). The groups successfully developed applications involving speech without further study of speech technologies or MASs but were unable to present their work in the context of MMD and could only provide theoretical discussion of systems architectures and processes such as fusion.
In phase-II, groups were given an introduction to MMD and an outline of the example architectures. They were also given instruction on writing fusion rules and the grammar/lexicon used by the language processor. The groups were required to configure the necessary sub-systems to process multimodal input (involving selecting suitable systems architectures, writing fusion rules and grammar rules) but they were not required to manage dialog. In contrast to Phase-I results, Phase-II students were more able to discuss system architectures from a practical perspective and appreciate the complexities of processes like fusion even though they had only used these (not built them).

The problems encountered by students were less concerned with the underlying architecture or the multimodal nature of the input than the difficulty of writing grammar and fusion rules to handle human-like language. The extent of this difficulty varied depending on whether the target group of students had some prior exposure to similar rule based systems.

Although more investigation is needed to judge its suitability for larger scale systems, both students and tutors commented positively that providing a suitable architecture had eased the burden of teaching and learning.

6 CONCLUSION

Our primary aim is to investigate how we can reduce the complexities of MAS and of MMD development in order to allow students to experiment practically with these systems. To achieve this we have developed an agent framework with a focus on usability and specified a generic architecture for MMD systems based on an API of agents for tasks such as speech, fusion and language processing. This allows students to experiment with systems at various levels of abstraction by building agents and by reconfiguring MAS architectures and their sub-components. While more discussion is needed within the community about the merits of different architectures and the suitability of different agent-based tools, the results of our work suggest that the barriers for wider development of MAS in education can be reduced.

7 REFERENCES

Allowing Trainers to Author their own Learning Content

Debbie Richards
Computing Department
Macquarie University
Sydney, Australia, 2019
+61298509567
richards@science.mq.edu.au

Meredith Taylor
Computing Department
Macquarie University
Sydney, Australia, 2019
+61298509065
mtaylor@science.mq.edu.au

John Porte
Computing Department
Macquarie University
Sydney, Australia, 2019
+61298509065
jporte@science.mq.edu.au

ABSTRACT

Computer Based Training can allow convenient and independent learning opportunities in the workplace and classroom. Developing these learning modules to contain content specific to an organisation, however, can be costly and beyond the reach of small companies. For the classroom teacher, development of modules is typically impractical due to time and skill requirements. For a learning environment inhabited by agents, we need a means by which the teacher/trainer can select appropriate agents and tell the agents what to do which also captures the appropriate domain, problem solving and pedagogical knowledge to be transferred to the student. We offer a simple and practical approach which allows training scenarios to be developed and knowledge to be acquired concurrently as the trainer interacts with the system and existing scenarios.

Categories and Subject Descriptors
I.2.1 [Applications and Expert Systems]; I.2.6 [Learning]; Knowledge acquisition; I.2.11 [Distributed Artificial Intelligence] intelligent agents; K.3.1 [Computer Uses in Education] Computer-assisted instruction (CAI)

General Terms
Design, Experimentation, Human Factors.

Keywords
Computer-based training, virtual reality, training simulation, content authoring.

1. INTRODUCTION

Computer based training environments promised a flexible and cost-effective method for learning new skills. Early prototype systems, such as SCHOLAR [2], GUIDON [3], STEAMER [6], the LISP Tutor [1] and SHERLOCK [8], have demonstrated the feasibility of computer-based training systems as well as pushing the boundaries of research in the areas of artificial intelligence, natural language processing, planning and user modeling. A more complete overview of the main concepts that have emerged from early intelligent tutoring systems is given in Wenger [18].

As computers have become cheaper and more powerful, computer-based training systems offer an interactive multimedia experience or possibly a complete immersive virtual environment to the user. Today, most large companies use some form of computer-based training system, albeit at a high development cost. The costs in time, technology and human resources associated with developing computer-based training resources are beyond all but the largest organisations. An alternative to custom development is purchase of off-the-shelf software. Commercial simulation environments tend to be expensive (licenses can be around 50K) and they tend to offer minimal user interaction, cannot be used to define complex training scenarios, and use scripted characters that cannot react appropriately to the user or the events.

Solutions to these problems are being sought by researchers. One key field of research is the use of intelligent agents and multi-agent systems. It is increasingly common for eLearning environments to include a pedagogical agent to perform one or more roles such as a tutor, peer or teacher. These agents have social ability based on the observation that learner’s perception of the learning experience is positively affected when a lifelike character is included in a computer-based interactive learning environment. This has become known as the persona effect [9]. Human qualities such as empathy [12], enthusiasm and interesting personalities [5] and expressiveness in terms of communication and levels of advice [9] have been perceived by learners and educators to be useful. A study by Moreno et al. [13] found that participant memory retention and knowledge transfer was better when the learner was assisted by a pedagogic agent compared to a computer-based text environment without the agent.

A number of agents (e.g. Carmen [10]; GRETA [15]) have been developed which differ in appearance, behaviour, intelligence, embodiment, believability, socially capability, and so on, according to the goals of the researchers and projects. Two very large scale initiatives in the area of building virtual environments populated with intelligent agents include Net Environment for Embodied Conversational Agents (NECA) [7] and Mission Rehearsal Exercise System (MRE) [18]. NECA started in 2001 supported by the European Commission in order to develop conversational agents that are able to speak and act like humans. MRE started in 1998 supported by the US Army in order to develop an immersive learning environment where the participants experience the sights, sounds and circumstances they encounter real-world scenarios.

As researchers involved with NECA and MRE would confirm, creating a high quality scenario is time-consuming and costly. A multidisciplinary team of specialists including psychologists, graphic artists, storytellers, cognitive scientists and computer specialists is needed. Given these resource requirements, it is not
feasible to create a comprehensive library of scenarios or even appropriate for research-oriented projects primarily seeking to create and test theories and develop a proof of concept. For the classroom teacher and most workplace trainers, the creation of virtual worlds for learning is beyond reality.

Despite the demonstrated benefits, a bottleneck in the widespread uptake of these technologies is the creation and development of learning modules. This is largely because it is not practical for design and development to be done by those who have the learning content, that is, the teachers/trainers. Currently, content and knowledge are created by a third party through a lengthy process involving document exchange, interviews, discussions and reviews. This process is hampered by the inherent difficulty of getting the expert to articulate what they know and then having the knowledge engineer or domain modeller (re)representing and encoding the input they gathered. Such a process is error-prone and thus validation needs to be carefully performed. Since acquisition and maintenance are difficult, there is pressure to get the content, flow, structure and associated rules and problem solving methods right. The burden is often placed on the intelligent agents or other components of the system to gain (or at least mimic) the deep understandings and years of experience held by the teacher or domain expert. This is a tall order.

An alternative is to share the burden of providing intelligence more evenly between the user and system. We propose an approach in which both parties have a light load because each party performs the role they are naturally good at. Simple agents are good at reacting and reasoning about what they know. Acquiring new knowledge and knowing what the student needs to know are much more difficult. The human teacher, on the other hand, knows what needs to be taught and what the learning outcomes should be. We want our pedagogic agent to incrementally learn this knowledge from the teacher as scenarios and learning content are developed directly by the teacher. Rather than the agent independently discovering the knowledge they need, we want the knowledge to be taught to the agent. The first approach is seen by some AI researchers as a holy grail; the second is achievable with current technology, keeps the human in the loop and uses AI as a support rather than a replacement for human expertise.

In our current project, we are particularly concerned with the creation of experiential simulation-type learning environments in which synthetic agents or characters inhabit a Virtual Environment and allow the student or trainee to see and hear (and feel) a number of possible situations. In the particular context of the domain we are working in, we are interested in providing a low-cost (in the sense of safe) training experience for custom’s and immigrations officers to learn about the detection of suspicious behaviour. Unlike many learning situations, we are interested in proving training experiences for situations where there is not necessarily a right or wrong answer and there is rarely any factual content that would need to be memorised. The goal is to learn by experience and trial and error, seeing if you can detect something that raises suspicion. Trial and error can be a costly way to learn (for the trainee, organisation and even society). By providing a computer-based virtual environment in which to try out multiple scenarios the trainee becomes exposed to more and more alternative situations.

As part of achieving the goal of affordability, we have used the Unreal Tournament (UT) game development toolkit to develop the Risk Management Mod (RMM) training system. We have also conducted other work creating learning material using The Elder Scrolls Construction Set (TESCS) by Bethesda Softworks. To create educational resources in UT requires considerable programming and technical skills. TESCS is less demanding but unless the lesson concerns fantasy characters and lands, its use in the classroom is limited. In both systems (and other game development toolkits), the ability to add domain specific knowledge or add intelligence to the characters is limited if possible at all.

In this paper we describe our prototype which demonstrates how trainers can create their own scenarios from scratch and modify existing scenarios to create an extensive library of scenarios. A key element of our approach is that knowledge regarding those scenarios is acquired through the process of scenario creation. In the next section we consider further the role of computers in training.

2. CONTENT AND KNOWLEDGE

To allow context-specific choices by the character and easy knowledge acquisition we have incorporated Ripple Down Rules (RDR) [4] into our approach. The use of cases and the RDR exception structure allows local-patching of knowledge which can be performed rapidly and easily by the user. By incorporating the RDR KA technique to a training environment the trainer is able to interact with scenarios and add new knowledge as exceptions by suggesting alternative outcomes and alternative input variables.

Training based on a story or scenario has to convey a series of points to the trainee. This is unlike computer games where freeform entertainment is the goal. We have to represent the general story we would like to tell and if we are going to give the user choices we have to allow the character in our story to be reactive and changes in the story line to occur. In some cases the user is immersed in the virtual world (e.g. Crawford’s Erasmatazz1), in other cases the world is more of a presentation (e.g. [11]). Both possibilities are available in interactive drama and will often depend on the situation and learning objective. For example in the case of Marsella, Johnson & LaBore [11] the human is an observer who can select from a limited number of possible alternatives within the drama so that dealing with the emotional issues raised does not become overwhelming. While we are interested in adding interactivity and choice into our system and studying the difference that these make on the learning achieved, our past studies (to be provided later) have shown that for learning purposes, interactivity is not necessarily preferable or even desirable when learning content is to be remembered and transferred. We also believe that choice is often provided to compensate for a small number of scenarios. By providing some choices the plot is seen to differ and thus it is hoped the user will become bored less quickly with the system. If it is easy to create numerous scenarios and variations, then allowing the user to pick alternative paths is not the main way in which variety is supported.

1 www.erasmatazz.com
Stories are traditionally structured linearly so that the author may first develop the plot, gain interest and build to one or more climactic events. Without a structured sequence of events the story may fail to keep the audiences attention and to convey the key ideas within the story. Swartout et al. [18] use a graph-based approach known as Storynet to allow deviations from the main story line while keeping control of the overall flow. Earlier work in ITS found that semantic nets could not provide procedural knowledge adequately and as found by Brachman and Levesque “inheritance of properties in conceptual hierarchies is a complex problem in general” [19, p.33]. According to script theory [17] people organise their knowledge in terms of scripts. Scripts are stereotypical sequences of events such as ordering food at a restaurant, attending a meeting, etc. However, knowledge is highly contextual and it is difficult for people to write scripts. Instead it is easy for people to retell a story or situation they have experienced. Also, it is difficult for people to articulate what they know outside of the situation in which they need to exercise their knowledge. The approach presented in the next section allows easy recreation of experiences (storytelling) and also allows knowledge to be captured during the retelling together with the ability to execute and modify the story and knowledge on the fly.

We have developed a prototype training simulation known as BOrder Security System (BOSS) using Vizard. The Vizard Engine is controlled remotely by a software environment we have created called SynthIDE which is controlled by Lua scripts, a language similar to Python. Our training simulation system includes several 3D animated scenarios that might occur in the domain of airport security. In previous versions of the system, all aspects of each scenario were hardcoded into a Lua script. Thus if a new scenario was needed, the author was required to have a basic understanding of programming and of Lua. As this is not a realistic expectation of customs staff, we created a scenario authoring system that allows any user to create a new scenario.
We describe next how agents and their attributes may be chosen. In our custom’s officer training domain we only allow the passenger agent to change, the trainee officer and supervisor could be changed in a similar manner but we see little value in the user changing these agents.

### 2.1.1 Changing the Passenger’s attributes

Clicking the button to choose a new passenger will display a new window with small pictures of the available avatars (Figure 1). Clicking on one of the small pictures will show an enlarged version of the selected avatar which the user can then select as the new passenger. Once the passenger has been selected, a new window will be displayed where the user can choose the passenger’s attributes (Figure 2).

The user also has the option to add a new field to the passenger’s attribute list (Figure 3) and specify whether they need to type in the value for the field or select from several values.

---

#### 2.1.2 Changing the Dialog

If the user wishes to change the characters’ dialog, they may do so irrespective of whether they choose a new passenger or not. Clicking the button to change the script will display a window with pictures of the avatars down one side. Clicking on an avatar will produce an empty text box at the end of the dialog in which a line of dialog may be written (Figure 5). Initially the window will contain the current script. The user may also add pauses in between characters speaking, delete a text box or move a text box up to change the order of the dialog. We can also add features which would allow the agent to perform certain actions such as “walk_to”, “snowball (moving hand in random circular motions)”, etc.

Once the user has completed their changes the code will once again be generated and they may save their changed Lua script. They may choose to update the original scenario or add the scenario as a new training situation. We plan to allow the addition of key words when scenarios are saved which would be associated with the scenario to assist with retrieval of scenarios as part of supporting development of a training library.

---

![Figure 3. Adding a new field to the passenger’s attributes](image)

![Figure 4. The newly generated code (in bold)](image)

---

![Figure 5. Changing the scenario’s script](image)
2.1.3 Merging with the Knowledge Based System

If a customs officer chooses to create a new scenario for training purposes, they most likely have in mind a new situation which presents a new risk and requires new rules of conduct to be learned by the trainees. For this reason we incorporated our Ripple Down Rule (RDR) knowledge based system into the scenario authoring system.

RDR [4] is a knowledge acquisition and representation technique which uses an exception structure to locally patch knowledge as it is found to be in error. The approach uses cases (or in this domain scenarios) to motivate the capture of knowledge. Similar to many case-based reasoners, RDR uses a fault-driven knowledge acquisition cycle involving: run new case against existing knowledge (in the form of production rule), if the user does not agree with the system assigned conclusion, they enter the correct conclusion and select features in the case which justify the new conclusion and also become the conditions in the new rule. The new rule is added as an exception to the rule that gave the misclassification. The new case is stored in association with the new rule and used in subsequent knowledge acquisition episodes to ensure that any exceptions to the new rule differentiate between the current case and the case associated with the rule to be overridden.

In keeping with this typical RDR knowledge acquisition cycle, after choosing new passenger attributes, the user will be shown the rules that the RDR system currently has in its database that correspond to the chosen attributes. The user can then either choose to agree with the rules currently in the RDR system, or create new ones (Figure 6). In this way, the user is not only creating a new scenario, but is also adding knowledge into the system about the correct course of action if such a scenario were to take place. Defining new rules for the scenario will also help the user to write the dialog since they have already established what the end result should be.

We are currently working on adding a button next to the delete and move buttons to allow the trainer to enter a rule in a more top down fashion if they feel there is specific reasoning behind why they have added/changed something in the scenario. For example, if the agent Officer is told to ask “Why don’t you have any luggage?” there may be an associated underlying rule such as:

\[
\text{If } \text{length_of_stay} > 2 \text{ days AND luggage = ‘none’ THEN risk = ‘high’ AND action = ‘ask why no luggage’}
\]

2.1.4 Direct Script Editing for Advanced Users

For more advanced users the scenario authoring system currently allows the user to change an existing Lua script by either editing it directly as an alternative to the more user-friendly approach described in sections 2.1.1 – 2.1.2.

At all times, the user is shown the script that they are editing. The parts of the script that are involved in changing the scenario are highlighted, and users can either click a button to specify the changes they want to make, or change the code themselves (see Figures 7 and 8 at the end of the paper). While it is not necessary to show the user the Lua code, changing the code directly is the fastest way to make changes, and some users may find they wish to do it that way. It will also give the user a better idea of how the system works. Currently the user may change the attributes and avatar of the passenger and the dialog of the characters.

Figure 6. Making a new rule for the scenario
3. FUTURE WORK

Currently our scenario authoring tool requires that users specify all aspects of the scenario which are then written into a Lua script. The dialog of the characters is not associated with any of the rules stored in the RDR knowledge base. For example if there is a scenario where the passenger has a criminal record for a major crime, it might trigger a rule that states:

\[
\text{If \ criminal\_record = 'yes' AND criminal\_offence = 'major',}
\text{THEN risk = 'medium' AND action = 'search passenger's bags'}
\]

This rule will then require the custom’s officer avatar to say “We are going to check your bags”; however, this line of dialog is not currently associated with this rule. Rather, the user authoring the scenario that triggers this rule must type this line of dialog in themselves which is then stored in the Lua script. The attributes of the scenario (such as the passenger’s details) are associated with the rule, but the scenario’s dialog is kept separate. This may not be practical in the long term as several scenarios may use similar lines of dialog even though they trigger different rules.

For example, a scenario may trigger the following rule:

\[
\text{If purpose\_for\_visit = 'unknown', THEN risk = 'medium' AND action = 'search passenger's bags'}
\]

The action required for this rule and thus for the scenario is the same as the action required for when the passenger has a criminal record for a major crime. We are therefore considering associating several predefined lines of dialog with various rules so that when a user is authoring a scenario, they may choose from a list of phrases related to their scenario to make their characters say. Thus a library of phrases will be built up with each phrase associated with one or several scenarios each of which is in turn associated with one or several rules in the RDR knowledge base.

We are also working on allowing users to dynamically alter scenarios while they are running in the virtual environment. Users will be able to pause a scenario while it is running and make changes to it before either running it from the beginning or allowing it to continue from where it was paused. This will give users a better idea of what their scenario will look and sound like as well as making them feel like they are interacting with the characters and environment.

Currently, the user can choose a new passenger from four 3D characters – two males and two females. We are working towards allowing the user to customise the passenger by choosing their age, hair colour and style, clothing and build to give the simulation a more interactive feel.

In order to develop a large repository of training scenarios we will implement a search engine to identify relevant scenarios to be used or modified. Also to support reuse we will be breaking scenarios into smaller components which can be grouped to form a complete scenario. For example, we have standard bag searching procedures, standard questions regarding filling in the incoming passenger card, and other dialogues which can be combined according to the specific situation. RDR will play a pivotal role in indexing these scenario snippets (sub-cases) so that indexing would occur as the snippet is created and avoid one of the problems experienced in case-based systems, the need to manually index stereotypical cases.

4. SUMMARY AND CONCLUSIONS

In this paper we have presented a scenario authoring system as part of a larger training simulation for risk assessment in the customs domain (e.g. [15]). Our scenario authoring system allows trainers to develop the script and properties of a scenario that may occur in an airport at the customs desk. We have also merged our RDR system with the scenario authoring system to allow the user to define new rules that should be applied to the scenario.

Currently our characters and environments lack realism in terms of facial expression, body gestures and movements, emotions, etc. While these factors are important in providing believable and embodied agents and such agents have been found to enhance the learning experience [5, 9, 13], we note that realism is not essential. Swartout et al. [19] were surprised by the willingness of the audience to suspend belief and overlook shortcomings in the simulation. They found that the degree of engagement of the observer which is affected by the story line and the believability of the actors will have a large impact on the observer’s satisfaction. We do not expect to provide a perfect environment but one sufficiently compelling to attract and keep the trainees attention so that they are willing to suspend disbelief and engage in the learning task.

The next stages in this project involve allowing users to dynamically alter scenarios while they are running and customise the 3D avatar of the passenger being questioned at customs. We are also working on linking predefined lines of dialog with rules in the RDR knowledge base so users can select some of the lines they wish to have in their dialog instead of having to write the whole script themselves.

To date we have completed a pilot evaluation of our approach to capturing knowledge as one interacts with a scenario involving 4 participants. Overall, the preliminary study yielded positive results on the usefulness of both the RDR system and the 3D training simulation. It also assisted us in targeting areas for further development in both systems. We expect to have a more comprehensive usability study completed in the next few months which incorporates the various technical, logistical and conceptual issues identified in the pilot.
6. REFERENCES


Figure 7. Lua script showing where users can choose a new passenger

```lua
--Start avatar
I_CreateAvatar("Officer1", "male10.cfg", "male10.dmd");
I_SendMessage("V_SetAvatarProperty", "Officer1", 
              
              
```

Figure 8. Lua script showing where users can change the script

```lua
I_SendMessage("V_AvatarExecute", 
```

```lua
I_SendMessage("V_AvatarExecute", "Passenger1", 
```

```lua
I_SendMessage("V_AvatarExecute", "Guard1", 
```

```lua
I_SendMessage("V_AvatarExecute", "Walker1", 
```
MAS Coursework Design in NetLogo

Ilias Sakellariou
University of Macedonia
Dept. of Applied Informatics
156 Egnatia str, Thessaloniki
54006, Greece
iliass@uom.gr

Petros Kefalas
CITY College
International Faculty
of the University of Sheffield
Dept. of Computer Science
13 Tsimiski str, Thessaloniki
54624, Greece
kefalas@city.academic.gr

Ioanna Stamatopoulou
CITY College
International Faculty
of the University of Sheffield
Dept. of Computer Science
13 Tsimiski str, Thessaloniki
54624, Greece
istamatopoulou@seerc.org

ABSTRACT

In the context of an Intelligent Agents course, we have chosen NetLogo as the means to satisfy the students’ demand for hands-on practice, to help them understand at a deeper level the otherwise theoretical aspects involved in the design of a multi-agent system (MAS). In this paper we present in detail the structure of the two pieces of coursework assigned to the students, the first one introducing students to the reactive architecture and the second, building on the first, to the hybrid architecture, also incorporating agent communication issues and interaction protocols. More particularly, we present an indicative MAS scenario that is given to the students as a case study for investigation as well as a thorough description of what is expected from them. The scenario facilitates practical agent design and simulation, contributes to the expected learning outcomes and provides various assessment opportunities.

1. INTRODUCTION

Setting an Intelligent Agent (IA) coursework assignment within an undergraduate one-semester course is a somewhat challenging task. On one hand, the nature of topics that are addressed by such a course, such as agent architectures, intentional notions and communication and cooperation protocols, dictate the need for offering some practical hands-on experience of the issues and problems involved in the area, in order to increase student understanding and facilitate learning. On the other hand, the time and student course load constraints imposed by standard undergraduate, and even some postgraduate, programmes could be prohibitive to employing one of the available fully fledged tools for agent development [1, 2, 3, 5, 9].

The challenge described above arose while teaching an Intelligent Agents course in the final semester of the final year of a Computer Science undergraduate programme. Out of a plethora of topics which are included in the course’s syllabus [13], it was considered important to provide practical experience in a number of fundamental aspects, such as representative agent architectures (such as reactive, BDI and hybrid), agent communication languages (such as KQML and FIPA ACL) and interaction protocols (such as the Contract Net protocol). The coursework assigned to the students had to fulfil three important learning outcomes of the overall course, according to which a student should be able to:

- discuss and synthesise agent solutions;
- sensibly design multi-agent systems;
- cope with key issues in implementing agent-based and multi-agent systems.

Given the constraints mentioned above, the adopted solution involved a multi-agent simulation platform, NetLogo [14], which in previous work we have appropriately extended by message passing and belief-desire-intention agent development libraries [12] so that certain restrictions imposed by the platform are overcome. The justification for choosing NetLogo over other options has also been reported elsewhere [13] and based on our class experience, it has proved to be quite a successful choice.

What we aim with the current paper is to present the approach that we have followed to coursework setting using NetLogo, justifying the choices we adopted and demonstrating further opportunities with respect to what can be assessed by similar assignments. In Section 2 discuss the rationale behind approach while Section 3 presents a taxi transportation scenario used as a basis for the coursework and, more particularly, how the environment has been modelled using the NetLogo programming structures. Sections 4 and 5 describe in detail the two pieces of coursework assigned to students, respectively, and Section 6 concludes the paper.

2. ASSESSMENT REQUIREMENTS

2.1 Assessment Rationale

As in any typical IA course, it was very important that the students gain some hands-on experience and that they are being assessed from a very practical perspective in:

- designing and evaluating a reactive multi-agent system for a given problem scenario,
- proposing and justifying the BDI and Hybrid architectures and precisely identifying components like beliefs, desires and intentions that are newly introduced in the context of the course,
• understanding and correctly using (in terms of semantics) the FIPA ACL agent communication language [6] in a MAS,

• successfully employing an interaction protocol in a specific problem scenario, and identifying all the issues that arise.

Resorting to a fully fledged platform was not a viable choice due to the constraints mentioned above: this was a final year course, in an already overloaded semester, in which students had to also work toward their final year project. Consequently, selecting an appropriate programming environment was a crucial issue, since the former had to adhere to a number of requirements, such as present the minimum installation problems, provide easy visualisation of the agent behaviour, support the multi-agent system aspects assessed and at the same time clearly demonstrate the difficulties in AMAS programming.

As aforementioned, the platform of choice was NetLogo [13], a modelling environment targeted to the simulation of multi-agent systems that involve a large number of agents. The platform aims to provide “a cross-platform multi-agent programmable modelling environment” [14]. A number of features make NetLogo an excellent platform for teaching IA [13]: a simple, expressive programming language with a small learning curve, rapid GUI creation and custom visualisations, an environment consisting of patches (components of a grid) and turtles (agents that “live” on the grid), enhanced through the use of user defined variables that allow the modelling of complex environments and agents with their own state, respectively. The platform directly supports the creation of reactive agent architectures, a feature we have taken advantage of not only for educational purposes but also as a means to support our research [10]. Communicating, reasoning agents (BDI or hybrid) on the other hand can be developed by using two libraries, specifically designed to support more complex multi-agent simulation on the platform [12].

As such, NetLogo together with the two libraries offers us the opportunity to set coursework assignments that meet our teaching needs.

2.2 Assignment Scenarios

Undoubtedly, the closer to reality an assignment scenario is, the more appealing it appears to the students. Such real world scenarios serve also the purpose of further establishing the fact that AMAS technology can potentially be applied to a number of areas. Throughout the years that the course has been delivered, we have used a variety of such scenarios, such as fire forest detection and prevention, airport logistics, satellite alignment, some of which have been reported in [11, 12, 13].

In order to further increase student interest and demonstrate how different architectures and protocols tackle the same problem, all coursework given to students in a cohort concern the same scenario. This imposes some constraints on the range of application areas that could serve as case studies but provides a better testbed for comparing various agent architectures. To summarise, in our opinion a good coursework scenario should:

• involve a real world problem, to which students can relate,

• be appropriate for both reactive and DBI multi-agent solutions, in order to meet our assessment requirements,

• have some sort of spatial reasoning that would provide a nice visualisation on the selected platform.

To illustrate our proposed approach to coursework setting, we describe in the sections that follow an assignment scenario concerning taxi transportation.

3. TAXI TRANSPORTATION SCENARIO

This scenario concerns taxi transportation of passengers located in various parts of a fictitious city to its airport. The idea is rather simple and common: passengers that require a ride to the local airport appear randomly in the city area. The term “randomly” refers both to the time and the location that the passengers appear in the city. Agents control taxis with the task of picking up the passengers and transporting them to the airport.

Obviously, the above problem can be easily tackled by both reactive agents, that randomly drive around the city looking for passengers and by BDI-hybrid agents that use a protocol to coordinate the transportation. Visualisation, on the other hand, involves creating a “bird’s-eye-view” outline of a simple city and allowing the agents to move on this city. The first issues that had to be addressed is how to model the environment and how to create a visualisation / experimentation environment that the students will use.

3.1 Designing the Environment

The NetLogo platform is ideal for rapidly creating such environments. First of all, the fact that a set of variables can be defined for each patch allows the modelling of complex environments. Furthermore, since turtles can inspect the patch variables, developing the agents’ sensors is greatly facilitated. In the specific case, setting the environment involves the following two issues:

• Modelling the streets, i.e. the set of patches where agents are allowed to drive. The decision was to include a patch specific variable road that gets the value 1 if the patch belongs to a road, 2 if the patch belongs to a junction and 0 otherwise.

• Providing information regarding the distance between a patch and an airport gate. This is useful in the case of reactive agents, since the latter rely only on local information in order to navigate toward the airport. The solution involves the introduction of a patch variable distance-airport that holds the Manhattan distance of the patch to the airport.

The environment is generated by appropriate NetLogo code, including the variable assignments required as mentioned above. Taxis, passengers, gates, streets and junctions are colour coded so as to provide immediate information on the status of the system to the user (Fig. 1).

3.2 Assumptions and Support

Students were provided with a set of procedures and a full GUI environment to run the experiment. The GUI controls allow the user to set the total number of taxis in the city,
their speed, a parameter that controls their random movement, and the total number of passengers that will appear during the experiment. A number of metrics and monitors were implemented including:

- the simulation time (ticks) required by the multi-agent system to complete the task,
- the number of collisions between taxis or of a taxi to a street edge, so that coursework assessment is facilitated,
- the number of passengers that have arrived to the airport, are waiting for a taxi or are on board taxis, and
- the number of passengers left to appear in the simulation according to the experiment settings.

The complete environment is shown in Fig. 2. Of course there is a number of other real-life features that could be implemented, such as traffic lights, other vehicles moving in the city, one-way streets, etc. thus a number of extensions to the actual environment itself could be implemented by the educator, or asked as part of the coursework by the students.

4. COURSEWORK 1: REACTIVE AGENTS

Based on our experience, students have much less problems understanding and applying reactive architectures, such as the subsumption architecture [4], for such scenarios. Thus the first coursework concerns the implementation of a multi-agent system consisting of reactive agents. According to the learning outcomes of the coursework the students should be able to:

- understand in depth the reactive agent architecture, its advantages and disadvantages,
- design a simple reactive agent to perform a task,
- build a simple prototype of a reactive agent system,
- evaluate the design choices made, based on the simulation results.

Students were assessed according to the following criteria:

- Correctness, originality and justification of the proposed agent architectures;
- Implementation and code documentation;
- Analysis and presentation of experimental results;
- Presentation of the report (clarity, structure etc.).

Since (a) students had not been exposed to NetLogo programming earlier in the curriculum and (b) the main aim was to provide an insight on the issues and problems regarding the reactive architectures, a suggestion is to release the NetLogo implementation of the environment and the agents’ sensors and actuators as part of the assignment handout. As a result, students can only concentrate on designing the reactive architecture and evaluating their design choices. The code fragment that follows is an example of the NetLogo code students were provided that concerns a sensor that detects a street edge on the left side of the agent:

```netlogo
; to-report detect-street-edge-left
  ifelse [road = 0] of patch-left-and-ahead 30 1
    [report true]
    [report false]
end
```

Note how easily such a sensor can be implemented, given the NetLogo primitives and the environment representation described in section 3.1. A set of agent actions was also released with the handout. The code that follows, shows the implementation of three taxi agent actions, the first one stochastically turning the agent by 90 degrees, the second one moving the agent ahead according to the speed set in the GUI and the third one placing the agent closer to the airport gate, by forcing it to face a patch with a minimum distance from the latter.

```netlogo
to turn-randomly-90
  let p random 100
  if p < probability-to-turn
    [ set heading heading + one-of [90 -90] ]
end
to move-ahead
  fd speed
end
to move-to-airport
  move-ahead
  face min-one-of neighbors4 with
    [road > 0] [distance-airport]
end
```

Given the set of agent’s sensors and actions students were asked to design, implement and justify a reactive architecture for the taxi agents. The architecture consists of a number of rules that dictate which is the appropriate action the agent should take, given the current input form the sensors. A simple inhibition relation relying on rule ordering is used and it can easily be ensured that at each execution cycle only one rule can fire. A model answer is shown below.
Figure 2: The scenario environment in NetLogo

Evaluation of the design choices was performed via the provided monitors, as part of the coursework. Students were able to evaluate their design based on the metrics mentioned above (time ticks, the total number of taxi collisions, etc). Incorporating other metrics such as the total waiting time of the passengers (or possibly the average), total distance travelled by the taxis etc., is a straightforward task. A number of other questions regarding experiment parameters were set for students, such as:

- How does speed affect the number collisions?
- What is the affect of the number of taxis to the overall system efficiency (time ticks)?
- What could be possible enhancements within the limits of the reactive architecture that could improve the overall system’s efficiency?

The first two questions require some experimentation and presentation of the results along with brief explanation and justification. The last question is more intriguing and might require further development of NetLogo code, thus allowing for a better assessment and mark distribution over the cohort.

5. COURSEWORK 2: HYBRID ARCHITECTURE, COOPERATIVE MAS

The second coursework concerns the design and implementation of hybrid cooperative agents and is far more demanding and challenging. The scenario in this case is as follows: each passenger can broadcast its request for transportation to all taxis, which in turn can reply to the request negatively or positively, in the latter case also reporting their distance from the calling passenger. Thus, both taxis and passengers are agents under this scenario.

Passengers are modelled as stationary and communicating BDI agents, with the top level persistent intention being that of “finding a taxi”. Taxis, on the other hand, since they have to move in a highly dynamic environment, i.e. in streets populated with other taxis, and navigate safely towards the airport, are modelled as hybrid agents: the lower layer is responsible for emergency action, such as avoiding collisions with other taxis and keeping the taxi within street limits, while the higher layer is responsible for message exchange, cooperation and plan generation.

The expected learning outcomes of the second coursework are that the students:
• understand in depth the issues and difficulties involved when building a multi-agent system, such as agent communication languages, interactions protocols etc.,
• propose a suitable agent architecture in order to perform a problem solving task,
• use an existing library to construct FIPA ACL-like messages and implement an interaction protocol,
• build a simple prototype of a multi-agent system,
• evaluate the design choices made, based on simulation results.

Students are assessed according to the following criteria:
• Correctness, originality and justification of the proposed agent architectures;
• Correctness and justification of the cooperation protocols proposed;
• Implementation and code documentation;
• Analysis and presentation of experimental results;
• Presentation of the report (clarity, structure etc.).

The assignment handout in this case included the implementation of the environment, which is identical to the previous coursework, the set of sensors and actuators of the agents (as in the case of reactive agents) and the two libraries [12] that extend the basic NetLogo platform, one allowing message exchange (message library) and the other the implementation of BDI agents (BDI library). This was considered necessary, since the standard NetLogo platform does not provide any primitives towards this direction (for a list of the primitives provided by the two libraries see Table 1). A simple example that demonstrates the use of the libraries was given to students in the form of a naive multi-agent system, where each passenger reports its position to all taxi agents and the latter rush to the caller with no coordination whatsoever. After identifying the problems with the given implementation, students were asked to:

• study and experiment with the given multi-agent scenario and propose and implement minor changes that could increase its performance (such as taxi agents not rushing to the first caller, but to the closest),
• proposing a cooperation protocol and defining the necessary FIPA messages that it dictates be exchanged,
• defining the beliefs and intentions of the agents under the proposed protocol,
• implement the system using the two libraries provided.

Naturally, the most appropriate solution involves employing the Contract Net protocol [7] between passengers and taxis. According to the protocol, passengers play the role of “managers”, while taxis assume the role of “contractors” and the evaluation criterion of bids is the distance of the taxi from the calling passenger.

The BDI library allows NetLogo agents to construct and follow plans of actions through gradual intention refinement.

<table>
<thead>
<tr>
<th>Manipulating Beliefs</th>
</tr>
</thead>
<tbody>
<tr>
<td>create-belief [b-type content]</td>
</tr>
<tr>
<td>belief-type [bel]</td>
</tr>
<tr>
<td>belief-content [bel]</td>
</tr>
<tr>
<td>add-belief [bel]</td>
</tr>
<tr>
<td>remove-belief [bel]</td>
</tr>
<tr>
<td>exists-belief [bel]</td>
</tr>
<tr>
<td>exist-beliefs-of-type [b-type]</td>
</tr>
<tr>
<td>beliefs-of-type [b-type]</td>
</tr>
<tr>
<td>get-belief [b-type]</td>
</tr>
<tr>
<td>read-first-belief-of-type [b-type]</td>
</tr>
<tr>
<td>update-belief [bel]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Manipulating Intentions</th>
</tr>
</thead>
<tbody>
<tr>
<td>execute-intentions</td>
</tr>
<tr>
<td>get-intention</td>
</tr>
<tr>
<td>intention-name [intention]</td>
</tr>
<tr>
<td>intention-done [intention]</td>
</tr>
<tr>
<td>remove-intention [intention]</td>
</tr>
<tr>
<td>add-intention [name done]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Message Exchange</th>
</tr>
</thead>
<tbody>
<tr>
<td>create-message [performative]</td>
</tr>
<tr>
<td>create-reply [performative msg]</td>
</tr>
<tr>
<td>add-sender [sender msg]</td>
</tr>
<tr>
<td>add-receiver [receiver msg]</td>
</tr>
<tr>
<td>add-multiple-receivers [receivers msg]</td>
</tr>
<tr>
<td>add-content [content msg]</td>
</tr>
<tr>
<td>to-report add [msg field value]</td>
</tr>
<tr>
<td>get-performative [msg]</td>
</tr>
<tr>
<td>get-sender [msg]</td>
</tr>
<tr>
<td>get-content [msg]</td>
</tr>
<tr>
<td>get-receivers [msg]</td>
</tr>
<tr>
<td>send [msg]</td>
</tr>
<tr>
<td>receive [msg]</td>
</tr>
<tr>
<td>get-message</td>
</tr>
<tr>
<td>remove-msg</td>
</tr>
<tr>
<td>broadcast-to [breed msg]</td>
</tr>
</tbody>
</table>

Although the library is rather limited compared to fully-fledge agent development systems, such as the PRS [8], it offers adequate facilities to develop agents of the level of complexity needed by such undergraduate coursework. For example in the code below, the top-level intention “find-a-taxi” is further refined to three lower level intentions:

```plaintext
;;; Plan to find a taxi (in reverse order)
find-a-taxi
    set color yellow
    add-intention "evaluate-proposals-and-send-replies" "true"
    add-intention "collect-proposals"
    timeout_expired cfp-deadline
    add-intention "send-cfp-to-agents" "true"
end
```

Table 1: Primitives provided to students of the FIPA ACL Message Passing and BDI NetLogo libraries
Note that the \texttt{add-intention} call adds an intention to a stack, and that the intention persists until its condition becomes true. The first argument of the \texttt{add-intention} procedure is the name of the intention that corresponds to a NetLogo user-defined procedure, while the second is the persistence checking condition. More details about the libraries can be found in [11].

The message library allows the implementation of all message exchange required by the cooperation protocol. For instance the following code sends the call for proposals to all taxis participating in the scenario:

```
[begin]
to send-cfp-to-agents
  broadcast-to taxis
  add-content
    (list "taxi needed" my-coordinates)
  create-message "cfp"
[term]
```

Taxis on the other hand are hybrid agents, since they need to respond immediately to emergency situations regardless of the overall plan the agent is following and at the same time allow the control to pass to the higher level if no emergency rises. Simple hybrid architectures can be implemented with ease in NetLogo, as shown in the code below:

```
[begin]
to taxi-behaviour
  ;; Reactive Layer
  if detect-taxi [turn-away stop]
  if detect-street-edge-left [rt 5 stop]
  if detect-street-edge-right [lt 5 stop]
  ;; Proactive Layer
  execute-intentions
[term]
```

The reactive layer consists of three rules that implement collision avoidance. If any of the rules fires, control does not proceed to the \texttt{execute-intentions} BDI library procedure, that “runs” the proactive layer of the agent. Thus, although the above approach is rather simplistic compared to the structures and models proposed for hybrid agents (red hybrids), it serves its purpose of demonstrating to students problems and notions involved in the design of such systems, and especially possible reactive - proactive layer interactions.

Taxi agents are initialised with the top level, persistent intention “listen to messages”, which is never removed from the intention stack and the following code extract shows the behaviour of the agent when it has adopted particular intention.

```
[begin]
to listen-to-messages
  let msg 0
  let performative 0
  while [not empty? incoming-queue] [
    set msg get-message
    set performative get-performative msg
    if performative = "cfp"
      [evaluate-and-reply-cfp msg]
    if performative = "accept-proposal"
      [plan-to-pickup-passenger msg stop]
    if performative = "reject-proposal"
      [do-nothing] ]
[term]
```

There are two interesting points in the code above. The first concerns message processing: agents can easily inspect the performatives of the FIPA like messages received and take appropriate action, by resorting to the primitives of the message exchange library. For instance, in this specific case, the \texttt{get-performative} primitive is used to extract the performative of the messages received. The second point concerns the complexity of the agent’s plans and is better demonstrated by the implementation of the following \texttt{plan-to-pickup-passenger} procedure:

```
[begin]
to plan-to-pickup-passenger [msg]
  let coords item 1 get-content msg
  let pass_no item 2 get-content msg
  let junction select-close-junction-point coords
  add-intention "move-to-dest " coords
  do-nothing
  add-intention "plan-to-pickup-passenger " pass_no
  [plan-to-pickup-passenger msg stop]
  execute-intentions
[term]
```

Through the execution of this procedure, the agent forms a plan to pick up and transport a passenger to the airport. This plan consists of navigation steps that move the taxi agent to the passenger location (first go to the closest to the passenger junction and then move to the passenger location), pick up the passenger and check that the passenger is on board, carry the passenger to the airport until you have reached the airport and finally drop the passenger. Note that, as previously, there are check points of the form of persistence conditions for the removal of an intention (the agent is committed to transporting the passenger to the airport until it has reached the airport), but also more elaborate check points in which the agent can revise its set of intentions —the \texttt{check-passenger-onboard} is such a case when the agent, if it has not successfully picked up the passenger, must remove the rest of the plan steps, i.e. its intentions, and inform the calling passenger with an appropriate “failure” message:

```
[begin]
to check-passenger-onboard [pass_no]
  ifelse onboard > 0
    [do-nothing]
    [remove-intention
      (list "carry-passenger-to-airport" "reached-airport")
      remove-intention
        (list "drop-passenger" "true")
      send add-content
        "sorry, I could not find you"
      add-receiver pass_no create-message "failure"
    ]
[term]
```

As demonstrated, by using the BDI library we can also
implement different commitment strategies and plan revising techniques and expose the student to the related notions and problems.

The opportunities to make the scenario more sophisticated are numerous and depend on the learning outcomes of the coursework that is to be set. Possible variations include:

- Introducing two types of taxi agents, some of which drive around the city picking up passengers reactively and the others being call taxi agents. This extension adds a bit to the complexity and since passengers could cancel a request, it demands a more elaborate interaction protocol.

- Introducing a calling centre in order to have passengers directing their requests to a single point that will allocate taxis to requests by initiating again a Contract NET protocol. This extension can become more interesting if there are more that one calling centres and the respective taxi agent teams compete.

- Allowing taxi agents to pick up more than one passengers on their way to the airport (Greek Taxi Transportation Scenario) and thus raise issues such as opportunistic planning, etc.

- Increasing the level of complexity of the environment, by introducing an addressing scheme for streets closer to reality (street names, street numbers, etc.), one way-streets and so on, so that the planning process of the agent requires more sophisticated techniques.

6. DISCUSSION AND CONCLUSIONS

Our intention with this paper was to disseminate our approach to the setting of coursework for an IA course and to share our experience with NetLogo assignments with colleagues who might be interested. What we have presented may either be used as a set of guidelines for setting the coursework or even as demonstration material for the purposes of clarifying the involved IA concepts in the minds of students.

We have followed the presented approach for a number of years now and our overall impression is that students enjoy this form of agent development, which, however limited, provides an insight to the issues raised when implementing Multi-Agent Systems. Their satisfaction increased in comparison to the early years of the introduction of the course in which coursework was restricted only to design issues and theoretical questions. We believe that both the visualisation environment and the minimum programming effort required contributed towards this direction. It should also be noted that their final examination performance has also been increased due to the better understanding of the intentional notions involved in MAS (beliefs, intentions, etc.).

Asides from the feedback we got from the students regarding their overall satisfaction, as part of general questionnaire they complete for all units at the end of every semester, and from the change we perceived in their performance, the previous academic year last we also provided a more targeted questionnaire, aiming to evaluate students’ perception particularly in relation to the use of NetLogo. According to this feedback, indicatively we mention that 93.8% of the students found that NetLogo’s visual environment helped them to better understand agent behaviour, and that the initial code provided for the 1st coursework assignment helped them in developing the solution. 71.4% of the students similarly felt that the provided BDI and FIPA-ACL libraries helped them in developing a solution for the 2nd coursework assignment. Finally, when asked, only 12.5% and 0% of the students would have preferred Java or Prolog, respectively, as the language of choice for their IA assignments (these are the two languages they are primarily exposed to in their studies). For the complete results of the evaluation, the interested reader is referred to [13].

The approach described in this paper, allows plenty of room for a number of issues and topics to be assessed. Agent planning, commitment strategies, agent architectures, message passing, cooperation protocol design and evaluation, issues on functional and spatial decomposition of problems, and even team formation and disbanding can be addressed given an appropriate scenario.

Future extensions include a number of issues. One of our first goals is to enhance debugging facilities with respect to the message passing and intentions. Currently, the user can view the intentions of each agent participating in the experiment by inspecting the corresponding agent variable, which is rather limited. All the messages exchanged can be displayed in the GUI environment, but given the high number of both participating agents and messages, their inspection requires some effort on behalf of the students. A solution could be to export both intentions and messages, appropriately time stamped (using ticks), and provide some animation/visualisation tool to facilitate debugging. Of course, we are currently investigating the implementation of various scenarios that involve other protocols, e.g. auctions, implementable with the use of the existing message exchange and BDI libraries. Finally, multi-team competition games, such as RoboSoccer, seem to be a natural extension of the current work, however, more enhancements to the standard NetLogo platform and existing libraries are required for such an environment.

7. REFERENCES


[8] M. P. Georgeff and A. L. Lansky. Reactive reasoning...


Teaching AI and IA to non-Science Graduates

Ioanna Stamatopoulou
CITY College
International Faculty of the University of Sheffield
Dept. of Computer Science
13 Tsimiski str, Thessaloniki
54624, Greece
istamatopoulou@seerc.org

Maria Fasli
University of Essex
School of Computer Science and Electronic Engineering
Wivenhoe Park, Colchester
CO4 3SQ, UK
mfasli@essex.ac.uk

Petros Kefalas
CITY College
International Faculty of the University of Sheffield
Dept. of Computer Science
13 Tsimiski str, Thessaloniki
54624, Greece
kefalas@city.academic.gr

ABSTRACT
As part of a Masters programme entitled Technology, Innovation and Entrepreneurship there was an apparent need to incorporate a course to introduce students to Artificial Intelligence (AI) and Intelligent Agents (IA). The reason was twofold: both AI and IA can be thought of as innovative technologies as well as technologies that can lead to the creation of innovative products. The task, however, of designing and delivering such a course turned out to be rather challenging due to various inherent restrictions, most important of which was the fact that the audience in its majority comprised of students of a non-science background. In this paper we present our approach to designing such a course, explaining how all restrictions have affected our choices, report our experience and evaluate the outcome.

Categories and Subject Descriptors
K.3.2 [Computer & Information Science Education]: Computer Science Education, Information Systems Education

Keywords
Multi-Agent Systems, Intelligent Agents, Artificial Intelligence, Teaching & Learning

1. INTRODUCTION
Teaching topics on Artificial Intelligence (AI) and Intelligent Agents (IA) to students with Computer Science-related background has long been recognised as presenting educators with a number of challenges. However, more recently and with the rapid expansion of the Internet and the World Wide Web, the innovations in business and the increasing impact of these technologies on almost every aspect of human life, AI and IA have become relevant topics to other disciplines too. Notable examples are Business and Management, where AI and IA could be seen as providing the means for innovation. Incorporating AI and IA in courses for non-specialised students presents educators with additional challenges regarding both content and delivery. One of the fundamental issues to address is that of the technical content involved: as non-specialist students do not have a program- ming background, such courses cannot be too technical as students will not be able to follow them. On the other hand, tackling such topics from a purely theoretical point of view can make the course seem dry, resulting in students losing their interest. Another issue that requires attention is that of relevance: how AI and IA are relevant, to business students for instance, needs to be well communicated and motivated so that they can comprehend the value of such a course. This is also pertinent to student engagement.

In 2008 we were faced with such a challenge; to deliver a course related to Artificial Intelligence, Intelligent Agents and Multi-Agent Systems to a postgraduate programme under certain conditions that diverge from the norm. The title of the programme is MSc in Technology, Innovation and Entrepreneurship or TIE [1]. The curriculum of this Masters programme includes: Knowledge Society and ICT Policy, Entrepreneurship and Innovation, Managing Strategic Change, ICT for Strategic Management, Managing Knowledge-Driven ICT Projects, Internetworked Business Enterprises, Innovation Management and New Product Development and Research Methods followed by a Dissertation. As is obvious from the curriculum, the programme contains a mixture of technology and business issues and aims to tie Technology, Innovation, and Entrepreneurship as the three key drivers of economic growth, provide an integrated, strategic view of management of technology and addresses the contemporary challenges general managers face today [2]. It is also obvious that the courses do not provide any room for technical skills to be acquired. The programme is aimed at non-Computer Science graduates who would like to acquire and enhance their knowledge and skills in Business Management, but with the focus being not on acquiring simply business management skills, but on understanding how innovation can be driven and managed with the purpose of creating agile businesses and organisations that can take the lead and remain competitive in a globalised economy. Finally, as this programme is aimed at graduates who may already be employed, the mode of delivery is in terms of short courses, i.e. each one of the courses is delivered over three days (Friday, Saturday, Sunday).

When designing the programme, its proposers suggested that there is room for a course that would include AI and IA, since they can be thought as innovative technologies as well as technologies which could lead to innovative products and give opportunities for entrepreneurship. There were,
however, restrictions for this course which are summarised as follows:

1. the course title should not contain the terms AI, IA or MAS;
2. it should be delivered over one long weekend (Friday, Saturday and Sunday);
3. it should be taught by a team of at least four lecturers;
4. the assessment should be done through coursework only;
5. the majority of the audience comprised of non-science students (e.g. business graduates).

Although our first reaction was that this task would be extremely difficult, taking into account the above restrictions, we agreed to undertake the challenge. In this paper, we would like to report our approach, thus aiming towards dissemination of practice to fellow educators that may face similar situations.

The rest of the paper is organised as follows. In the following section we describe in detail the design of the course in terms of its aims, learning outcomes, content etc. Section 3 discusses issues and practices relevant to the coherence of a course that expands over a number of related but independent sessions, and to the coordination that has to be achieved between all members of the teaching team. We present in Section 4 our approach to motivating the students on the course and in Section 5 some further details on our attempt to engage them. Section 6 provides a description of the coursework assigned to the students and assessment rationale and in Section 7 we discuss our experience and present student evaluation results. Section 8, finally, concludes this paper.

2. DESIGN OF THE COURSE

2.1 Title of the course

One of the issues that we were faced with was that of naming the course (restriction 1). Firstly, we wanted to avoid using the terms AI or IA in the course’s title as there were concerns that the non-specialist students attending this programme would find it intimidating. Secondly, we felt that the inclusion of these terms might be misleading as we did not intend to provide a technical exposition of the topics covered. Thirdly, we wanted a title that would be able to capture the context of, applications and uses of AI and IA in driving innovation and successful entrepreneurship forward. Having discussed many alternative titles, we reached an agreement for “Knowledge Technologies for Innovation”, KTI for short, so that it is apparent that the course is a review of various knowledge technologies with the focus placed on how they can contribute to advancing products and services in a corporate environment.

2.2 Course description

In a rapidly changing world the path towards innovation demands efficient storage/retrieval and exploitation of knowledge, a “new” business asset that aims to increase the performance of classic enterprise systems and facilitate quality decision-making. The course provides an introduction to knowledge technologies, with emphasis in representation techniques and engineering methodologies. It offers an overview of knowledge-based systems and intelligent agents together with a series of case studies to clearly demonstrate their applicability and potential for innovation. The Semantic Web, an important research area with a potentially large amount of business applications is also discussed. The course presents areas of AI and IA, whose application can significantly increase the efficiency of existing processes in an enterprise.

2.3 Aims and Learning Outcomes

We found restriction 5 to be the most challenging one, which actually drove the entire design of the learning outcomes, content and, of course, teaching and assessment. As stated in the syllabus of the course the aim is to introduce Knowledge Technologies, Artificial Intelligence and Intelligent Agents as an innovative way towards entrepreneurship and as means for innovative products, software and services in the business world.

By the end of this course, the students should be able to:

- Discuss both theoretical and practical issues regarding knowledge representation and engineering.
- Understand the principles of knowledge based systems, with a special emphasis on how the offered technology presents an opportunity for innovation.
- Provide an overview of the intelligent systems technology and its applications in various aspects of enterprises.
- Discuss the importance of the Semantic Web and the related emergent technologies, such as semantic web services, ontologies and agents, and illustrate the crucial role they will play in future systems’ interoperability.
- Illustrate through the study of real-world cases the importance of the application of innovative artificial intelligence technologies and trends.

2.4 Duration

Restriction 2 could not be relaxed due to the nature of the programme which is delivered only through long weekends, one for each course. Teaching is spread over 2.5 days throughout which 12 sessions of 2 teaching hours each are scheduled (see below for content). As a consequence, restriction 3 requires the teaching to be distributed among at least four different people (in our case five), with no more than 8 hours in total allocated to one person. One is appointed as a course leader with the main responsibility to oversee the organisation of the course as a whole, compile a teaching team and monitor the coherence of its teaching.

The total learning hours of the course amount to 150. Excluding the 24 hours of teaching throughout the long weekend, the remaining learning hours involve structured e-learning activities (on-line collaboration with course leader, self-assessment test, forum etc.), self-study and preparation of the assessed coursework, spread throughout one month following the delivery of the material.
2.5 Textbooks and Reading Material

Unfortunately, there is not a single textbook that could satisfy all requirements for such a course. As it is normally the case in postgraduate courses, the reading material was based around a set of known textbooks [5, 7, 8, 9] and a set of published papers.

2.6 Content

It was rather optimistic to be able to cover all areas in 24 teaching hours. Bearing in mind restriction 5 and that we should skip any technical design and implementation details, we have decided to compromise depth in favour of breadth. Briefly the sessions provided by the course are the following:

- **S1** Introduction to Knowledge Technologies
- **S2** Introduction to Artificial Intelligence
- **S3** Knowledge Acquisition, Representation & Knowledge-based Systems
- **S4** Ontologies
- **S5** Semantic Web
- **S6** Semantic Web Services
- **S7** Introduction to Intelligent Agents
- **S8** Agents for E-Commerce
- **S9** Advanced Knowledge Technologies I
- **S10** Advanced Knowledge Technologies II
- **S11** Principles in Robotics
- **S12** Conclusion and Discussion

More detailed descriptions of the contents of each session can be found in Table 1.

2.7 Other arrangements

The course documentation, i.e. syllabus, information regarding the teaching team, schedule and contents of each session, presentations, links to other related sites, videos and demos, coursework assignment etc., is given to students ahead of the teaching weekend in a CD, preferably a month before. All material is also uploaded in advance to the Learning Management System of the college.

3. COORDINATION AND COHERENCE

As mentioned above, restriction 2 and its consequent restriction 3, implied that good coordination and detailed preparation of the course is required in order to achieve consistency and coherence between sessions. As such, the tasks that the course leader undertakes are the following:

- design of the syllabus,
- compilation of a teaching team,
- discussing the topics to be presented,
- overseeing the preparation of teaching material,
- discussing and designing a coursework assignment,
- monitoring the running of the teaching weekend,
- contact with the students (on/off-line and face-to-face),
- teaching, particularly Sessions 1 and 12 (first and last),
- assessing the coursework submissions, and
- evaluating and reviewing the course.

The extra tasks, in this case, in comparison to those that any lecturer undertakes, are to discuss the topics to be presented with each individual in the teaching team and to oversee the preparation of teaching materials by each member of the teaching team. The coordination required is delicate and the discussions require subtle handling in order to achieve consistency and coherence without causing anxiety or raising conflicts between colleagues. To enable the teaching team to coordinate the coverage of topics, avoid overlaps and facilitate the smooth transition from one topic to the other we agreed that for each session the corresponding member of the teaching team bearing in mind the overall syllabus of the course, should prepare a one-page document providing:

- a brief description of the session,
- a list of the covered topics,
- the session’s learning outcomes,
- the relevant reading material, and
- WWW and multimedia resources.

The course leader is then responsible to aggregate the learning outcomes, compare them with the overall learning outcomes and adjust the session descriptions accordingly. The main difficulty is to maintain a uniform balance between depth and breadth across all sessions and of course eliminate duplication of topics covered between sessions. The interested reader may refer to [3] for all necessary details on all sessions.

4. MOTIVATION - CAPTURING THE STUDENTS’ INTEREST

As mentioned in the introduction, one of the issues when designing such a course for non-specialist students is that of relevance. Hence, another important challenge for us was to present the students with a motivating example that would enable them to see the potential of using knowledge technologies to drive innovation and support entrepreneurship. This example would also enable the members of the teaching team to focus on specific theoretical issues and practical applications. The initial thought was to adopt existing scenarios to attract the interest particularly of the non-science students. Such scenarios, for instance, have been developed by AgentLink Special Interest Groups in order to focus consideration of, and provide a context for, technological advances needed for future systems [6]. We soon realised, however, that it would be better to devise our own scenario, so that it:

- covers all the areas we would like to talk about in the course,
- presents some state of the art technology,
<table>
<thead>
<tr>
<th>Session</th>
<th>Description / Contents</th>
</tr>
</thead>
</table>
| S1: Introduction to Knowledge Technolo-
| gies                                | Innovative Knowledge Technologies, Areas covered in this unit and initial definitions, Success Stories (business, commerce, science, everyday life)                                                                                                                   |
| S2: Introduction to Artificial Intelli-
| gence                              | Brief history and future of AI, Basic problem solving by state space search, Implications to software development and software products, Implication to business world                                                                                       |
| S3: Knowledge Acquisition, Representa-
| tion & Knowledge-based Systems      | Data, Information & Knowledge, Reasoning, Knowledge Acquisition, Knowledge Representation, Knowledge-based Systems                                                                                                           |
| S4: Ontologies                       | The importance of semantics, particularly for system interoperability, Key elements of ontologies, Stages in ontology development, Languages for developing ontologies, The Web Ontology Language (OWL)                       |
| S5: Semantic Web                     | Brief historical overview of the web, Technological limitations of the current Web, Semantic Web structure and individual layers including technologies                                                                                                     |
| S6: Semantic Web Services            | Service Oriented Computing and Service Oriented Architecture, Core Web service technology standards: WSDL, SOAP and UDDI, Shortcomings of current Web service technologies and challenges still to be addressed, The case for knowledge technologies: automation of activities throughout the Web service lifecycle, State of the art in the area of SWS: different frameworks and applications |
| S7: Introduction to Intelligent Agents| Historical overview, Definitions and characteristics of agents and multi-agent systems, Differences between agents and other related technologies as well as synergies, Interactions in multi-agent systems: communication, cooperation and negotiation |
| S8: Agents for E-Commerce            | The impact of e-commerce to individuals and organisations, The potential uses of agent technology in e-commerce based on the Consumer Buying Behaviour model, Shopbots, current technologies, limitations and future, Agents in markets behaving strategically, The concept of a dominant strategy in protocols |
| S9: Advanced Knowledge Technologies I | Knowledge Extraction from Databases (Data Mining) processes and applications, Machine Learning: principles and types of learning                                                                                                   |
| S10: Advanced Knowledge Technologies II| Planning & Scheduling, Neural Networks as a machine learning technique, Natural Language Understanding, Vision                                                                                                                |
| S11: Principles in Robotics          | Robotic basics, Problems and applications, Success Stories (business, commerce, science, everyday life)                                                                                                               |
| S12: Conclusion and Discussion       | Forum for discussion, Summary of the course, Coursework assignment, Assessment criteria                                                                                                                                 |

**Table 1: Brief descriptions of course’s sessions**

- gives motivating examples for AI and IA related products,
- relates to some existing products we could demonstrate, and
- sounds like a science fiction scenario in order to engage the students and provoke discussion.

The scenario, called “Just an ordinary day”, is the following:

Thursday evening. I stayed at the office late but there are a few things to do before I call it a day. I had an interview with a prospective student today and she requested whether Computer Science or Business Studies was more appropriate for her. I type: “Look at Anna Papadopoulou’s profile. She applied yesterday. Give me probabilities of successful completion of studies, based on her high school record, with special attention to maths, English and essay writing”. I get a positive response and an estimate of time for the result. It is rather late, I am tired and can’t wait to go.

I request to log off and there comes a pop up: “You haven’t checked your calendar during the last five hours. Would you like me to remind you what your schedule is tomorrow?”. I’d rather not. I know there is nothing urgent. “Is there anything else you want me to do?”, it insists. As a matter of fact, yes there is! I just realised I haven’t made any travel arrangements for a conference. I type: “Search for best value tickets for Venice, Italy. Fly out 2 December, fly Back 5 or 6 December. Advise for hotels, giving priority to those listed in the Informatics Education Conference 2008 web site. Pay registration fee. Use visa. Deadline is Monday”.

I step out of the office and start walking to my car. My mobile beeps me a couple of times to notify me that there are friends around the corner. I ignore it. The thing is that I do not know where my car is parked because Panos, my husband, used it and left it somewhere close, but where? I take out my mobile and locate my car —it is in the parking lot nearby. I reach the car and put my fingers on the handle. The door opens and I place myself on the driver’s seat. I touch the steering wheel. My seat is adjusted to my height and so do the internal and external side mirrors. I remember that when I bought this car, it took it a couple of weeks to learn my preferences. My thumb starts the ignition. The car drives me to the exit. While settling my things, I listen...
to a car message: “The back left passenger window does not operate. The problem seems to be on the central control switch.” The bar of the parking lot opens well in advanced allowing me to get on with some decent speed. I then take control of the car. The radio is on. “Search for Scorpions. Song with ‘let me take you far away’ in it”, I say. A voice replies: “Found one song: 'Holiday'”. “Play it!”, I confirm. “Downloading and playing” is displayed on the screen. In a second or two the song starts to play, but a short while later the volume goes down and I am warned: “There is an incoming message to your mobile! Hear it now or later?”. “now”, I say. “Message from home refrigerator: No milk”. I don’t need that right now! “Give me directions to the closest open supermarket, preferably avoiding traffic”, I plead.

While I am getting into the supermarket, I remember we have a shopping list. “Call home!” I ask. There is no reply, however. “Connect to RobSpy!” I instruct. RobSpy is one of my latest bargains. A small tripod with a camera on board, connected to the net, aimed for kids surveillance at home. “RobSpy connected!”, I am actually seeing the inside of my house real time. I instruct RobSpy to go to the kitchen and look at the fridge as this is where we post our shopping list. In a minute I have the shopping list displayed on my screen’s mobile. While shopping my trolley’s screen displays the total amount due so far. Passing through the cashier is a matter of seconds.

I go back to the car. “Hold all messages!”, I instruct. At last, I can listen to my music. I had a busy day, but thanks to my assistants I saved some energy for tomorrow.

The first action we took after writing the first draft of the scenario was to distribute it to the teaching team together with the Introductory Session slides, so that all of us have a specific reference of what we should aim to demonstrate to students. It worked out very well and the teaching team appreciated the scenario which gave them some guidance on which issues to focus on given the tight restrictions mentioned above.

5. DELIVERY

The way we designed the schedule and delivery of the material through the 12 sessions imposed another serious restriction; ideally all students should be present and engage in active participation and discussion in the introductory session. Otherwise, we thought that the course would fail to deliver what was intended. Thus the scenario was the first thing presented in Session 1 and we provoked a discussion by asking the following questions:

- Is the scenario real or science fiction?
- Which parts are real and which are science fiction?
- Is there technology to support it?
- Is there innovation in these products / technology?
- What are the problems (technological, business) that prevent it from being a reality?
- Why do such products not exist in abundance in the market?
- Are there entrepreneurial opportunities in this scenario?

Following the discussion, which went as expected (most of the audience believed in the ‘science fiction’ part of the story), we revisited the scenario, this time by stabbing the main points of interest. We analysed the examples given by referring to the related technology and the definition that accompanies it and cross referenced to the relative session. However, further action was required to change the student perception towards the non-science fiction side. We collected a number of videos and applications which demonstrated that most of the products exist in one form or another. Indicatively, the following is a list of videos and sites, that we presented them with:

- a fingerprint reader,
- a face recognition software or a smiling shot camera,
- natural language understanding demos, successes but also failures,
- speech synthesis,
- commercial robots,
- autonomous vehicles, such as those participating in the DARPA challenge and commercially available parking assistants,
- various location services,
- intelligent super market checkout,
- intelligent e-shops,
- trading agents,
- scripted intelligent agents,

and others along the same lines. Links to all of the above for the interested reader have been collected and can be found in [4].

A similar pattern is followed in the rest of the sessions [3], with motivating examples leading the discussion, thus providing the opportunity to present the underlying technology and the challenges for innovation and entrepreneurship.

6. COURSEWORK ASSIGNMENT

As stated above, the assessment of the course is made only through coursework. Students were expected to submit a 3,000 word report in which they have to identify existing IT products or services and employ their critical and innovative thinking as to how these can be extended/infused with the knowledge technologies introduced during the course, so that they become innovative and more competitive in the business arena. More particularly, the assignment is divided into two parts:

- Identification and evaluation of four different existing products/services that can benefit from different existing knowledge technologies.
- Analysis relating to the most promising of the identified products/services of how the suggested knowledge technology is to be incorporated into it.

The assessment criteria briefly are the following:
It was interesting to find out, when assessing the assignments, how diverse the students’ perceptions and ideas are. At this point we should mention that they had been instructed to keep their ideas “small and simple” so as to avoid having to go into deep technical detail, as this has not been the aim of the course. As expected, a few of the ideas belonged to one of the extremes, either being too simple and not really involving AI or the technologies taught, or being too ambitious and large-scale, attempting to incorporate the majority of the technologies (e.g. a learning distributed knowledge-based system holding medical records and assisting in the diagnosis and treatment of patients across hospitals). In their majority, however, the students performed quite well, managing to properly identify potentials and propose well-grounded ideas, a lot of which were centred around the use of intelligent agents in combination with the semantic Web (e.g. a shopping assistant that allows a user to search for prices of products and possible points of purchase in his/her vicinity, a recruitment assistance agent for matching job seekers to prospective employers etc.).

7. DISCUSSION

The original requirements imposed for the design and delivery of the course Knowledge Technologies for Innovation seemed to be rather restricting. However, we believe that the way the course was delivered managed to satisfy most of them and that it has been beneficial both to the students as well as to the teaching team. Students were asked to evaluate the course using a standard evaluation form and a Likert point scale (1...7). The categories scored are Transfer of Knowledge, Presentation and Delivery, Motivation and Interest, Relevance and Usefulness for all sessions as well as all members of the teaching team individually, and Overall Satisfaction for the course. The results (average of all lecturers) are shown in Table 3. It was noted that the differences between sessions and lecturers is insignificant, something which indicates that we have achieved the desired consistency and coherence. We were very happy with the results, bearing in mind that such a course is not the most popular among non-science students. The questionnaire was answered by 15 students, as this was the first time the course run, and of course further validation of the outcomes is needed in the future.

In their comments, students mentioned that they found the motivating scenario and the videos very useful. More particularly, (quoting the students’ comments) they found the “material excellent and well-structured”, the “examples very useful” and the entire course to be “very motivating and inspiring”, “providing food for thought” and “very coordinated as all presenters knew exactly what the others had presented”, the latter proving that our approach to coordinating the design of all sessions and all of us using the scenario as a point of reference was fruitful. Not surprisingly, the students enjoyed the last session on robotics and the demos of real inexpensive commercial robots, like i-SOBOT, RoboSapien V2 and Lego Mindstorms NXT. On the other hand, there was one comment on the “knowledge being very technical”, requesting that the course discusses “even more business opportunities, approaches and aspects” and another commenting that the material “is a little condensed and complex but motivating”. Since achieving a balance between the inherent technical aspects involved and the bigger picture was our biggest consideration when designing the course, we consider it a success that the number of students who found the course technical and not enough business-oriented is restricted to two.

The members of the teaching team found the course really challenging and we believe that we have achieved the original learning outcomes. Additionally, it has been very interesting to learn how students with a business background perceive AI and IA and what ideas they have for entrepreneurial opportunities.

We were also given a chance to gather more audio-visual material for our mainstream Computer Science courses in which we also used the motivation scenario with success.

8. CONCLUSIONS

Teaching AI and IA can be a challenging task even within a pure Computer Science curriculum, mainly due to the breadth and diversity of the related topics, and the technical issues involved. Thus we were faced with even greater a challenge when required to design and deliver such a course as part of a Masters programme addressed to students of varying backgrounds. As the course has only been delivered once, we are definite that aspects of it can be further improved, especially considering that this has been a learning experience on our part and gave us the opportunity to familiarise ourselves with different perspectives that people of a business background have. Our experience, however, has further supported our initial belief that there is a need for such a course when educating the entrepreneurs of tomorrow, as technological innovation, which is so closely linked with these scientific fields, is a key factor to success and competitiveness. Exposing non-science students to such technologies is crucial, so long as it is done in a balanced manner that is abstractive enough to be understandable and appreciated, and does not overburden with the technical aspects. It is our belief that with this first attempt, we have managed to fulfil the intended aims and learning outcomes of the course, a belief which is additionally sustained by our students.
9. REFERENCES


Re-conceiving Introductory Computer Science Curricula through Agent-Based Modeling

Forrest Stonedahl, Michelle Wilkerson-Jerde, Uri Wilensky
Center for Connected Learning and Computer-Based Modeling
Northwestern University, Evanston, Illinois, U.S.A.
{forrest, m-wilkerson, uri}@northwestern.edu

ABSTRACT
We present a preliminary version of the MAICS (Multi-Agent Introduction to Computer Science) framework, which is a new approach for revitalizing introductory undergraduate or high school computer science curricula through the deep integration of agent-based modeling (ABM) and multi-agent systems (MAS) perspectives. We have developed a suite of educational agent-based models highlighting several key ideas of computer science. We discuss the merits of using multi-agent systems as a lens for conceptual understanding across disciplines, and how this approach can be beneficial for exploring topics that computer science educators might not normally consider to fall under the heading of ABM or MAS. We show that this perspective offers insights for many sub-fields, including searching, sorting, optimization, graphics, machine learning, networks/security, and operating systems. In particular, we highlight several areas where parallel, distributed, stochastic, and emergent methods can be incorporated fruitfully into early computer science curricula that too often focus solely on serial, deterministic, and centralized algorithms. It is our belief that an ABM/MAS paradigm can also improve accessibility of content for students, by providing motivating example models, and a ‘glass-box’ approach that encourages both understanding and experimentation. Furthermore, bringing disparate topics in computer science together through the common focus on emergent systems can promote a broader, more accurate, view of the field as a whole.

Categories and Subject Descriptors: K.3.2 [Computing Milieux]: Computer and Information Science Education – curriculum.

General Terms: Human Factors

Keywords: Computer Science Education, Agent-Based Modeling, Multi-Agent Systems, Curricular Models, Computational Thinking

1. MOTIVATION
Last year, Rick Rashid, a senior vice president for research at Microsoft, asked the rhetorical question of whether computer science is a dying profession [29]. Indeed, shrinking undergraduate computer science enrollment and concern about the underrepresentation of both women and minorities in computer science has been the subject of much speculation, concern, and debate, particularly in North America [18, 22, 20]. Diversifying the introductory curriculum is one approach for reaching a broader audience (see, e.g., [18, 17, 19]), which has met with some success. In this paper, we present the MAICS (Multi-Agent Introduction to Computer Science) framework as a new and powerful method for diversifying the introductory computer science curriculum. Through the MAICS framework, we demonstrate the potential to address many conventional topics of computer science (such as searching, sorting, optimization, graphics, machine learning, networks/security) in an unconventional way, through an agent-based modeling (ABM) and multi-agent systems (MAS) perspective. The framework focuses on two central goals. First, it seeks to enrich early (“low-level”) computer science courses by engaging students with conceptually rich “high-level” topics. Second, it emphasizes the role of distributed, decentralized systems; a concept that has strong implications both within and far beyond the domain of computer science.

More specifically, the first strand of our research aims to address the enrichment of early (“lower level”) computer science courses with a series of dynamic agent-based models coupled with compelling and interactive visualization. While an ABM/MAS approach can easily reach out to interdisciplinary examples from fields such as biology, economics, physics, sociology, biomedicine, and others, for the purposes of this paper we chose to stay primarily within the bounds of computer science. This approach provides the additional benefit of offering a survey of several conventionally “higher-level” topics, and gives introductory students a broader intellectual taste of what computer science has to offer, beyond “hello world”, sorting algorithms and syntax errors. Concepts of elementary programming can be covered concurrently through experimentation with the provided source code, and through extending the model or writing new agent-based models from scratch.

The second strand is based more on the technical content of computer science education and our vision of the future of computing. In recent years, computing and computer science have been undergoing an important shift toward parallelism. This includes the current prevalence of multiple and multi-core processors, the ubiquity of high performance computing clusters in academia and industry alike, cloud computing, massive peer-to-peer networks, social networking and Web 2.0 applications, increased deployment of massively parallel supercomputers for research, consumer-grade GPUs (graphical processing units) that deliver a teraflop of parallel computing power, as well as parallel languages and language features to accompany these developments. We
are not advocating that CS101 students need to be learning GPGPU programming techniques or dealing with mutual exclusion semaphores for accessing shared memory. The point is a broader one: that it is time to re-examine whether the prevalent focus in contemporary introductory computer science courses on centralized, deterministic, serial algorithms is best preparing our students in the long run, when they will eventually need to face a world of computation that is ubiquitously distributed, potentially stochastic, and increasingly parallel. Our work is further motivated by the larger goal of providing universal instruction in “computational thinking” (as described by Wing [37]), wherein students across all disciplines become fluent in computational methods and models. The MAICS framework is a step in this direction, both in supporting a broader view of computer science, and working to increase the accessibility of computational concepts for a larger audience.

Toward addressing these two concerns, we have developed a suite of educational agent-based models highlighting several key ideas of computer science, which illustrate the ideas of the MAICS framework. Since the word “agent” connotes different meanings to different audiences, we should mention that we interpret “agent” somewhat broadly to include very simple elements acting with a small fixed set of rules, and we do not limit its use only to agents that use highly sophisticated decision-making processes. Furthermore, the agents used in the suite of models we present here are all fairly simple agents, which we believe is appropriate for an introductory course on computer science, and also beneficial for developing skills in decentralized thinking.

The paper is structured as follows. We first discuss related research in computer science education, and argue for the merits of using agent-based modeling (ABM) as a “lens for conceptual understanding” when exploring topics that computer science educators might not traditionally consider to be in the domain of ABM or MAS. Next, we explore three example curricular models in some detail, and discuss how they may be used to promote understanding of multi-agent systems while learning about computer science topics. We also offer a brief overview of each of the other models in the suite. We conclude with some remarks about potential implementation considerations and discussion of our future work.

2. RELATED WORK

There have been many suggested approaches for revitalizing computer science education. In this brief review we list only a few, for comparison to our own ideas and approach. Specifically, we believe that introducing agent-based modeling to introductory computer science curriculum addresses many calls in the computer science education literature to engage students in motivating consequential tasks and to highlight the interdisciplinary nature of computer science and its applications. Additionally, we believe that the ABM/MAS paradigm is particularly amenable to introducing computer science topics at not only the collegiate level, but also to students at the secondary level and earlier.

Some proposed changes to help promote understanding, motivation, and retention in the introductory computer science sequence include the integration of design-first programming [29], pair programming [36], and robotics [11]. Others argue that attention to content delivery techniques should complement efforts to promote students’ own relationship

with computer science material: Naps et al. [26] discuss the role of visualization in CS courses, and in particular argue that “[visualization] technology, no matter how well it is designed, is of little educational value unless it engages learners in an active learning activity.” We argue that agent-based modeling offers an excellent approach for addressing these issues. Specifically, a curriculum designed around the ideas of ABM/MAS can provide an effective coupling of advances in computer science education methods (e.g., visualization technology) with the more general goals of active engagement and intellectual inquiry on the part of students.

In addition to local changes to the core computer science curriculum, many have argued for computer science to be better integrated into a broad curriculum centered around science and technology. In response to declining CS enrollment, Denning and McGettrick [18] call for a “recentering” of computer science, lamenting that in the public mind “computer science” has become narrowly associated with the job of “programmer”, and suggest an introductory CS sequence with a theme of technological “innovation.” Our suggestion to use ABM/MAS as an introductory theme is a less radical change, particularly since learning the art of computer programming remains a central piece of our educational framework (students will simply be programming multi-agent simulations, rather than, for example, writing programs for counting prime numbers). However, we do agree that early computer science courses often provide too narrow a view of what it means to be a computer scientist, and suggest that our approach will offer broader exposure to “upper level” topics.

Cushing et al. [17] suggested broadening introductory CS by offering interdisciplinary courses with math and science (such as ecology) as a means to improve retention and increase interest in the field. We believe that the ABM/MAS paradigm is especially conducive to such interdisciplinary integration, and one particularly powerful interdisciplinary idea is that of emergence – how the interactions between agents each exhibiting simple behavior at the individual level can result in surprising and complex aggregate-level phenomena that appears to be “more than the sum of its parts.” [21, 30] Emergence is a key conceptual bridge to a multitude of disciplines (including chemistry [23], materials science [12], electromagnetism [32], and biology [35]), and we believe it is equally important for it to be an ingredient in computer science education. In Section 3 we illustrate how emergence is a key component of many of the models in the MAICS suite. With the MAICS framework we have chosen to focus on topics that are considered within the discipline of computer science in order to provide a broad survey of higher level computer science topics, but at the same time the ABM/MAS paradigm allows us to make connections to a wide array of interdisciplinary endeavors.

This notion of core concepts within and between disciplines is also why we refer to ABM/MAS as a “lens for conceptual understanding” in computer science. Often, subjects such as neural networks, particle systems, genetic algorithms, sorting and searching, are organized topically within the computer science curriculum, and are thus taught separately, without making conceptual connections between them. In contrast, an ABM paradigm uses concepts such as agents and micro/macro level phenomena in order to highlight the similarities and differences between the mechanisms at work, rather than focusing primarily on the subject area.
Finally, while most introductory sequences in computer science are offered at the college or university level and we present this framework primarily in that context, we also acknowledge the important role of computer science education at the pre-collegiate level [28]. Just as in college, enrollment in high school computer science courses is low, and there have been calls for a more diverse, integrated computer science curriculum [20]. We suggest that an ABM/MAS paradigm, and the MAICS framework specifically, is also a powerful way to introduce computer science to a younger audience. In support of this assertion, we note that the NetLogo modeling environment [34], and even several of the agent-based models discussed specifically in this paper, have been successfully used in workshops and educational interventions as early as primary school (as well as with industry professionals and academic researchers both familiar and unfamiliar with computer programming).

3. THE MAICS FRAMEWORK

The MAICS framework is situated to address several goals, including the enrichment of early CS courses with a broader range of content, improving students' understanding of parallel, non-deterministic, and distributed systems, and offering a more exciting and dynamic introduction to the field. We have developed a wide collection of agent-based models (available for download from the NetLogo Models Library http://ccl.northwestern.edu/netlogo/models/), and for the purposes of this paper and the MAICS framework we selected a cross-section of those models that relate to important or motivating topics in computer science. The suite of models we describe herein consists of nine models spanning seven topics, as shown in Table 1. Some of these models have been used with great success in short workshops, or introductory courses on multi-agent modeling, but we have yet to implement an entire course using this approach. We present here a cohesive framework for such a design that we hope will be refined through trial, as well as feedback from others working in this area.

This is certainly not intended to be a comprehensive list of topics in computer science that could benefit from re-examination from an agent-based perspective. Instead, we seek to highlight several examples where parallel, distributed, stochastic, and emergent methods can be incorporated fruitfully into early computer science curricula that too often focus solely on serial, deterministic, and centralized algorithms. Furthermore, this list contains only fully implemented and documented models that are presently ready for educational use. More models could certainly be added to this list, highlighting other important ideas. Some of these topics (such as searching and sorting) are similar to those traditionally covered in an introductory CS sequence while others (such as particle swarm optimization) are more typically found in upper-level undergraduate or even graduate-level courses. Due to space constraints, we will discuss only the first three example models in detail, and then briefly explain the scope and purpose of each of the others.

These models were all implemented using the NetLogo [34] agent-based language and integrated modeling environment, which permits interactive modification of the model's parameters and the code itself. The NetLogo language, following the Logo tradition [27], has also been designed to be easy to read and easy to learn, and the integrated modeling environment contributes to a low barrier for entry [33]. Equally important, NetLogo is no “toy language”; it is a real language currently being used by researchers across the globe, offering a wide range of control structures and data types, and it is extensible via the Java programming language if access to additional libraries is required. In addition, NetLogo's built-in facilities for model visualization provide students with a convenient graphics library. In short, even if our curriculum was not based around a multi-agent perspective (which is NetLogo's predominant feature), NetLogo would still be a suitable choice for a first course in computer programming. We also wish to emphasize the “glass box” nature of the suite of models: besides the visual interfaces (shown in figures below), each model comes complete with educational documentation and full source code that students can easily edit and run within the NetLogo modeling environment.

3.1 Searching: PageRank model

Traditional computer science curricula invariably include discussions of searching, often starting with students learning to do a sequential search in an array of numbers of strings. Later on, they are often taught how to perform a binary search of sorted data, and to search other data structures such as trees or graphs, perhaps using depth-

---

Table 1: MAICS suite models and related computer science topics, listed by order of appearance in this paper.

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageRank</td>
<td>Searching</td>
</tr>
<tr>
<td>Painted Desert Challenge</td>
<td>Sorting</td>
</tr>
<tr>
<td>Virus on a Network</td>
<td>Network Security</td>
</tr>
<tr>
<td>Simple Genetic Algorithm</td>
<td>Optimization</td>
</tr>
<tr>
<td>Particle Swarm Optimization</td>
<td>Optimization</td>
</tr>
<tr>
<td>Artificial Neural Net</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>Particle Systems Flame</td>
<td>Computer Graphics</td>
</tr>
<tr>
<td>Flocking 3D</td>
<td>Computer Graphics</td>
</tr>
<tr>
<td>Dining Philosophers</td>
<td>Operating Systems</td>
</tr>
</tbody>
</table>

Figure 1: A screenshot from the PageRank model. Larger nodes represent higher PageRanks.
first search, breadth-first search (or perhaps Dijkstra’s algorithm). While we have no desire to debate the merit of these venerable and classic algorithms, we note that they are all designed to run deterministically on a single processor accessing an unchanging data set. It seems prudent to balance this with a decentralized algorithm designed for searching massive quantities of constantly changing data, i.e. the World Wide Web. Furthermore, we suspect students may be more motivated to learn about how Google “magically” returns relevant search results about their favorite curling team, as opposed to discovering how to find the position of “milk” in an alphabetized grocery list. The PageRank model [6] (see Figure 1) is based on the now famous PageRank algorithm developed by the founders of the Google search engine in the late 1990s [14]. PageRank is not technically a search algorithm, but rather a ranking algorithm, which provides a basis for ranking the information on one page as being more useful/important/relevant than the information on another page. The algorithm assigns a PageRank score to each web page, based on its relationship to other pages determined by the hyperlink structure of the web. Our PageRank model actually demonstrates two distinct agent-based methods for calculating the PageRank of a directed network (such as the web), though the two methods result in the same limiting behavior, and ultimately would assign the same PageRank scores to each page.

Method 1: Random Web Surfers. In this case, we assume there are “page” agents which are connected to each other in a directed network of hyperlinks, and there are also “web surfer” agents, which operate using these simple rules. They start at a random web page, and begin wandering the web. To wander, they either click on a link from the current page and travel to a new page, or they may (through some unspecified means – perhaps a TV commercial, typing in a web address, an email from a friend, etc) jump directly to a random page somewhere on the web. If they run into a dead end page, they also jump to a random page. The probability with which they follow a link versus jump to a random page is controlled by a parameter called “damping factor” (typically set at 85% chance of link-following). As these agents, move, the model records the number of times a web surfer has visited each page. One definition for the PageRank metric is given by the probability of a single random web surfer being at that page at a given instant. Using the random web surfers model, this can be easily calculated by dividing the number of visits for each page by the total number of visits. In more formal mathematical terminology, this can be viewed as finding the stationary distribution for a certain Markov Chain, where each page is a state, and there are transitional probabilities specified between each pair of states. However, introductory CS students do not need to have acquired this level of mathematical formalism to appreciate the emerging behavior of the agent-based model.

Method 2: Diffusion of PageRank Scores. In this case, the primary agents in the model are the web pages themselves. Each page starts off with an equal amount of PageRank score. At each time step, pages divide their PageRank up equally, and send it off as a gift to each the web page that they link to. (Pages with no out-bound hyperlinks are treated as if they linked to every single other page in the web.) Each page then receives PageRank gifts from each of the pages that link to it. Also, each page receives a certain amount of PageRank, just for existing (determined by the “damping-factor” parameter). This redistribution of PageRank via diffusion is carried out repeatedly, and over time the PageRanks converge toward the correct PageRank value. Mathematically, this method is related to the “power method” for finding the dominant eigenvector of a modified adjacency matrix for the directed graph formed by the hyperlinks.

Beyond the clear benefits of exploring and understanding this classic algorithm that is so instrumental in making information accessible on the web, our PageRank model also provides an excellent launching point for students to experiment by creating their own distributed link analysis and/or ranking algorithms. For example, students could endow the “random surfer” agents with more sophisticated behavior (use of the “back” button, bookmarks) and see how the rankings would be affected. A broader discussion about emergent search techniques could also encompass ant foraging mechanisms, or the search of fitness landscapes performed by genetic algorithms (making a connection to the Simple Genetic Algorithm model also included in our suite).

3.2 Sorting: Painted Desert Challenge model

Sorting algorithms are another staple of early computer science education, inevitably including at least several of the following collection: bubble sort, selection sort, insertion sort, merge sort, quick sort, heap sort, bucket sort, shell sort, and radix sort. Again, a common theme is the deterministic single-threaded and serial aspects of sorting (although many of these algorithms can be at least partially parallelized). As a counterpoint, we wish to present a messier, distributed, and stochastic view of sorting, in the Painted Desert Challenge model [7]. While it may strike some as an incredibly inefficient approach to sorting, one should note that it is intrinsically parallel, reasonably robust, and could be applied in situations where the data is shifting during the sorting process, as a result of noise. However, it is important to keep in mind that we are not interested here in arguing for the merits of this particular sorting algorithm, but instead we are arguing for the merits of the ideas that students will be exposed to by exploring this model. The Painted Desert Challenge model offers insight into emergent systems, and in particular ant colony and other problem solving techniques inspired by nature.
Rather than focusing on lower-level details of security, such as open ports or overrun buffer exploits, the Virus on a Network model [3] (see Figure 3) is concerned with security on a grander scale. In particular, worms and viruses that self-propagate from computer to computer through the Internet form a grave risk for today’s society due in part to the creation of large “botnets” capable of acting in unison to carry out destructive distributed denial-of-service attacks, or other illicit activities. Virus on a Network is an abstract model, based on the SIR (Susceptible, Immune, Resistant) models found in epidemiology. The setup consists of nodes (i.e. computers) on a network, and links between them, which could represent a variety of different connections depending on the attack vector of the virus (e.g., email contacts, shared network drives, shared USB keys, external hard drives, or floppy disks, etc). Nodes start as susceptible, except for some specified number that are infected with the virus. With some probability (which is controlled by an adjustable model parameter), a node that is infected by the virus can spread that virus to each of its neighboring nodes. Infected nodes also have a chance of recovering (e.g., an antivirus program removed the virus but didn’t close up the vulnerability), and they have a chance of recovering and becoming resistant to future attacks (e.g., an antivirus program inoculated the computer against this virus).

Through exploration of the model, students can learn about how the parameters affect the rapidity with which the virus moves through the network, as well as the lifetime of the virus, and the extent to which vaccination of a few nodes can or cannot prevent a widespread epidemic. The Virus on a Network model also has potential connections to other disciplines (such as medicine, marketing, or sociology), and promotes high-level discussions about computer security practices, the structure of social and computer networks, and the Internet. This also leads naturally to student projects and extensions of the model. For instance, the default network structure found in this model is based on spatial proximity of the nodes, with nodes that are closer together in the 2D plane having a high probability of being linked, whereas there are no long-distance links. Students can discuss whether such a configuration is plausible for virus contagion1 and write code to generate other types of network. A few other possible extensions include allowing the virus to mutate and evolve, and thus be able to reinfect computers which had become immune to a previous version of the virus, or to allow for coordinated (botnet) attacks by groups of infected nodes, or two have multiple different viruses present in the network. There are always opportunities for ambitious students to take this type of work further, and spin it off into summer research projects.

3.4 Remaining Model Suite Overview

The remaining six models in our model suite (shown in Figure 4) cover topics from an additional four areas of computer science. The Simple Genetic Algorithm model [5] and the Particle Swarm Optimization model [4] both offer an introduction to the area of stochastic optimization algorithms. While these topics are not usually covered un-

---

1Generally speaking, it is not. Real-world networks usually display a power-law degree distribution and “small-world” structure where long-distance links are present. We purposely chose this spatially-restricted structure to benefit visualization of the processes at work.
Figure 4: Model screenshots. Top row: Simple Genetic Algorithm, Particle Swarm Optimization, Artificial Neural Network. Bottom row: Particle System Flame, Flocking 3D, Dining Philosophers.

...thoroughly accessible to introductory-level students through the ABM/MAS perspective that is being cultivated through this framework. Additionally, the brevity of the code and the visualizations included in the models enhance the accessibility of content for these topics. Both of these examples show how very simple agents, acting with limited intelligence and information, can result in a population of agents moving toward a goal. In the Simple Genetic Algorithm model, this progress is measured by plotting the fitness and diversity levels in the population over time, as well as showing a visual representation of the best individual agent solution found in each generation. In the Particle Swarm Optimization, the progress towards a goal can be viewed as agents traverse a 2D fitness landscape, searching for a global optimum. These two models also offer interdisciplinary connections to evolutionary biology and particle physics. Artificial neural networks are also amenable to elucidation through an agent-based perspective, as we hope to demonstrate to students through exploration of the Artificial Neural Net model [2]. Each perceptron can be conceived as an agent, which follows certain rules during the training phase, and then another set of rules when it is being tested. This is another fairly advanced topic, which admittedly may take some effort for students to understand and appreciate. However, we should remind the reader that it is not necessary for students to understand every detail of the back-propagation training algorithm or what the nice mathematical properties of a sigmoid function are – such things can wait. The important thing is for students to gain a qualitative understanding of how the agents are activating each other, and that by automatically modifying the weights of connections between agents, it is possible for the system as a whole to “learn” pattern recognition skills. A class side-discussion comparing and contrasting this agent-based model with the neural networks found in humans should also prove provocative and educational. Agent-based modeling is useful in computer graphics as well, and is being increasingly explored as a means of automatically creating realistic procedural animations of systems with many interacting creatures or objects. Through the Particle System models[1] students can get a taste of the classic “particle systems” approach sometimes used in cinematic animation to create the illusion of water, fire, or smoke. While each particle is fairly passive, being pushed or pulled by an externally determined force field, it is still useful to think of each particle as one agent of a distributed multi-agent system, and it is not difficult to modify the code to make the agents take a more active and/or intelligent role in their movement patterns. For instance, more sophisticated agent behavior is exhibited in the “Boids” algorithm [31] for creating realistic-looking flocks of animated creatures, which is the inspiration for our Flocking model [8]. As the Flocking screenshot in Figure 4 shows, the NetLogo modeling environment also has a 3D version that allows development and visualization of models in three dimensions, which provides students with an early glimpse into programming in 3D environments as they work to extend, modify, or create their own multi-agent computer...
4. DISCUSSION AND FUTURE WORK

Our intention here is to offer a window into an alternative introduction to computer science. In practice, we would not expect this approach to be used to the complete exclusion of other curricula. We emphasize that in many cases it would be most beneficial to compare and contrast centralized and decentralized approaches to the same topic. Furthermore, the introductory course can still have a strong emphasis on learning to write computer programs. However, starting with existing programs in this case, agent-based models provides an opportunity for students to explore, to modify, and to learn to read the language at the same time as they learn to write it.

One important thing to note is that the MAICS framework, and an ABM/MAS approach in general, still allows for the integration of a number of other techniques shown to be beneficial for computer science education. There is no reason, for example, that the pair programming approach [16] or the integration of robotics [11] cannot be implemented successfully within the context of MAICS. Indeed, the NetLogo programming environment includes interfaces to various physical devices and a variety of "bifocal modeling" [13] activities, which allow users in a variety of contexts to compare computational agent-based models with real-world data collected using robotic sensors and actuators. The VBot curriculum [10], designed primarily for middle school students, engages users in programming independent robot agents, which can then interact with one another in a shared context (such as a robot soccer arena).

Several avenues present themselves for future work. One interesting consequence of integrating ABM and MAS into introductory level computer science courses is that students are encouraged to think about multiple programming language paradigms. Specifically they are encouraged to consider how centralized and decentralized systems can be used to solve different problems, or even to solve the same problems differently. This is exciting in light of recent work that shows that even students with no formal computer science training are able to reason about concurrency, and some even to develop both decentralized and centralized solutions for problems involving concurrent systems [24]. However, the questions of when and how students should be introduced to multiple paradigms (e.g., agent-based programming, object-oriented programming, functional programming) in the computer science sequence deserves further attention. While teaching a wide range of paradigms in early courses offers students with a variety of choices to better solve different problems, there is the danger of leaving students adrift in a sea of shallowly understood paradigms. Additionally, implementing the MAICS framework would provide a context for additional cognitively motivated research about the affordances and constraints of decentralized thinking for students with regard to the principle ideas of computer science.

Another key piece of future work is to perform a concrete implementation of this framework in the form of an introductory level computer science course, or two-course sequence. We suggest that an implementation will provide empirical support for the theoretical underpinnings of the MAICS framework, as well as offer new insights about the potential for introducing students to computer science through an ABM/MAS paradigm. NetLogo and NetLogo models have been successfully introduced to and used by novice programmers in a variety of domains, and we expect that doing so in the context of computer science education using the MAICS framework will produce similar results. Specifically, we would evaluate how this implementation affects the major goals: (a) student engagement and retention rates, (b) student ability to exhibit distributed thinking when presented with problems that can benefit from it, (c) student knowledge of programming fundamentals (such as basic control structures, recursion, loops, variables, etc.). Previously, we have evaluated both student engagement and distributed thinking skills with positive results, but this was in the context of undergraduate and pre-collegiate computer science courses and workshops that focused on modeling and simulation, rather than as a first course on computer science. In preparation for this work, we encourage discussion regarding implementation concerns and potential pitfalls for integrating agent-based modeling into computer science curriculum.

Through the MAICS framework we are offering a first attempt at producing a coherent introductory computer science curriculum centered around a series of agent-based models spanning a variety of topics. We believe that this framework addresses recent calls by computer science educators to introduce widely applicable, engaging curricula early in the computer science sequence that focus on the notion of "computational thinking", rather than specific algorithms and techniques. There unquestionably remains much room for improvement in this framework, and we hope that feedback on the selection of appropriate models or multi-agent systems that highlight important areas of computer science, as well as a broader discussion of the ideas we have proposed, will enable us to further refine these ideas.

Acknowledgments: We are particularly grateful to our colleagues – past and present – at the Center for Connected Learning and Computer-Based Modeling for their contributions to the models and ideas presented in this paper.

5. REFERENCES

Model References:


General References:


http://ccl.northwestern.edu/netlogo.
Short Papers
Crafting a Personalized Agent-Oriented Mobile E-Learning Platform for Adaptive Third Level Education

Olapeju Latifat Ayoola  
School of Computer Science & Informatics, University College Dublin, Belfield, Dublin 4, Ireland.  
+353-1-7162486  
olapeju.ayoola@ucd.ie

Dr Eleni Mangina Phelan  
School of Computer Science & Informatics, University College Dublin, Belfield, Dublin 4, Ireland.  
+353-1-7162858  
eleni.mangina@ucd.ie

ABSTRACT
This paper presents an agent-based m-Learning platform, “Mobile E-learning Platform”, which incorporates personalisation and collaborative learning for adaptive e-learning. The main objective of this platform is to provide University College Dublin with a single supported intelligent mobile learning environment that will promote adaptive and collaborative learning, human computer interaction on mobile clients anywhere, anytime and also to provide useful recommendation about available educational resources.

Categories and Subject Descriptors

General Terms
Design

Keywords
E-Learning, M-Learning, Mobile Technologies, Multi-Agent System, Collaborative Filtering, Ubiquitous Computing, Adaptive Personalisation.

1. INTRODUCTION
Distance learning systems and e-learning has become a great challenge and focus of many research teams in recent years. Educational tools and resource contents’ support are now provided for technical requirements and at affordable prices [1]. This evolution has given distant learning towards standard learning environments. E-learning systems are very popular in Information and Communication Technology due to the improved adaptivity and e-courses’ personalisation requirements [7]. The internet is the medium for distributing content more efficiently anywhere, anytime; Content can be presented with reduced costs for administrative tasks [12]. Traditional educational systems are mainly based on client-server or peer-to-peer architectures, which have shortcomings such as poor scalability and low availability. Another approach such as service-oriented encapsulates e-learning contents [8, 11, 17] inside a Web service to improve interoperability and reusability [10]. This approach benefits educational systems; contents are made available on the Web in different formats and the Web services can provide functionality that extracts and present them [11].

Personalisation is another shortcoming of the current e-learning systems (ELS) and one of the strong claims of the work presented in this paper. In the previously years, some ELS introduced useful suggestions to help students’ lesson sessions, monitoring student’s progression and considering students’ learning styles [4]. An extracted suggestion can be content-based, recommending a similar course based on students’ browsing history or search query. This collaborative filtering is carried out by searching for similarities between students and recommending similar modules [7]. University College of Dublin (UCD) has a modularized, credit-based educational system known as UCD Horizon that provides adaptive learning. UCD’s fast and efficient wireless local area network (WLAN) provides access to resources anywhere, anytime for its vast scale of students. This offers great opportunity for mobile clients’ users and e-learning facility. UCD has Managed Learning Environments (MLEs) that’s a resource repository and also an intelligent learning environment that aids students through their learning stages. These MLEs lack personalisation, efficiency, and interoperability. Majority of the services the MLEs provide, such as collaborative learning, are redundant because tutors and students are not making use of them. The skills learnt to use one MLE is not transferable to other mobile learning environments. [3] Existing MLEs are designed for clients such as desktop computers and notebooks hence it’s not ubiquitous enough.

2. PERSONALISED AGENT-BASED MOBILE E-LEARNING PLATFORM
The Mobile E-learning Platform (MEP) is designed to embody the qualities of managed learning environments, personalization, content management and agent systems. The system will be developed with Java, Java Agent Development (JADE) Framework, Lucene’s API, high-level scripting language, PHP, and markup languages, HTML. MEP has an agent-based service-oriented architecture that coordinates learning-related activities and content management such as features and learning materials. MEP will provide services via opera mini and built-in web browsers for clients such as PDA, mobile phone, media players, game console. MEP will also be accessible by Firefox, internet explorer, safari, Google chrome and other browsers for laptops and desktops. The mobile clients and computers will access MEP through a wired or wireless internet connection (Fig. 1) in a ubiquitous manner for an enhanced ease of access.
basic M-Learning management tool (Fig. 2) focuses on basic content delivery and support services. There is an intense focus on context-aware design and delivery because of the limitations that faces mobile clients such as power durability, screen resolution, and connection bandwidth. The advanced and persistent development of mobile devices in its very competitive market has immensely enhanced personalization and content delivery. A lot of e-learning tools make use of advanced methodologies such as collaboration filtering, adaptive navigation support for providing enhanced learning experience. In order to enhance the learning experience in UCD, MEP is designed with fusion of Lucene API, JADE, adaptive navigation support and content management to provide learning resources and personalized content delivery anywhere, anytime for all UCD students. In comparison to all related systems [2, 5, 6, 9, 14, 15, 18], MEP is a distinctive, intelligent managed learning environment that provides resources and personalized content to users anywhere anytime with the dynamic blend of different technologies to enhance teaching and learning experience in 3rd level education.

The architecture of the proposed system aims to address UCD’s MLE’s deficiencies such as lack of a mechanism to interoperate with other learning systems and interchange information such as educational material; lack of personalization (user’s features are not used for personalization) and lack of fusion of technologies, methods and principles for the provision of personalized e-education to meet the requirements of most users. The architecture will encourage knowledge sharing and content-reuse via collaborative learning and social networking; create cliques of students who have common interests via social networking; encourages personalized information delivery; provides an efficient personalized content recommendation in regards to user’s interest and academic strength; provides a complete service solution for third level e-educational system and provide context-aware delivery for mobile clients’ and PC users in a ubiquitous manner.

Figure 3 illustrates the general components of the proposed MEP. MEP services will be accessed via web pages hence during learning session, students’ devices such as mobile phone can access MEP directly via a web browser. Students are required to authenticate before accessing MEP services. The system provides registration service, which the students need to use to acquire login access. The login system will store user’s session while user is browsing the services, once user logs out the session is ended. In order to reduce privacy issues, the user sessions will not be used for collaborative filtering.

The system will detect the device which the student is using to access its services. At login stage after authentication, student can adapt screen to mobile view - if it's detected that they are using a mobile client - but this is optional to students since most 3G devices support full webpage view. Hence it depends on the mobile client the student is using and the student’s preference. After logging in, student can search for courses, attend study group session and enroll for study group session.

Anytime student logins in the personalisation service also observe the tools not of use. It makes use of this observation by suggesting to student to either remove the tools or keep it. It will also promote redundant features that are useful to student by giving then a brief explanation of the tools’ usefulness. The content management model is in charge of helping the student to find courses, book and attend studying sessions. Ontology model collects the search terms, edit it and retrieve keywords. It also checks the type of searching type, i.e. users or courses, the user specified. It sends the keywords and the preferred search type to the Lucene API. Lucene searches through index content and pass results to the agent platform. The agent platform now refines the search results by re-ranking the result in order to user’s learning strength and interest which are provided by the content register and profile register. The refined results will be sent to content management which now request for personalization according to the user’s preference, e.g. display five results at a time.

The authoring model will enable lecturers or tutors to sending updates or announcements, edit, publish and create learning contents. The tool will enable students to update, create, edit their study groups, profile, discussion boards etc. Learning content are learning materials, learning services such as groups, etc. When groups, discussion board and discussion topics are created, its details are stored inside the database. These details can be retrieved by the content registry and passed onto the agent platform. The agent-based [16] platform will be able to access other platforms such as iUMELA [19] to provide access for students to pay fees, enrol for courses, etc. Content and Profile
Registry will retrieve details about users’ interests and courses from the database. The content manager will also support interoperability between any of the services. Hence personalization manager, the progress monitor and accounting manager can collaborate on student interaction with the system and progression of the learning sessions. It will also provide the services such as chat tools, mailing, accounting, progress monitor, group tools, discussion board tools and any other services.

A progress manager will coordinate all assessment result for online examinations, specific module assignments’ results and general assignments’ results into an overview. An overall module overview of a specific module will be available to both tutors and students while the overall modules overview will not be accessible for tutors. An accounting manager will manage study group enrolment and store student participation for system analysis. Student’s identity will not be stored, only the number of students that employ the service will be stored. Profile manager will be in charge of creating student and module profile (i.e., user and document modeling) for collaboration filtering.

2.1 Functionality

In order to demonstrate the functionality of the system, we can consider a new student, who will register to the system with UCD email address and create a password. To access the system, students will go through authentication process which will verify their registration. The system (Fig. 1) will detect the device students’ use to access its service and offer options of mobile view or normal web page view if the device is a mobile client. Once a student is logged in, he or she can access learning contents and MEP services. Students can search for suitable modules, similar students, etc; enrol to the modules; receive updates for modules, etc; retrieve course material; utilise e-learning tools (i.e. online tutorials); attend a class (i.e. online lessons); submit assignments and take online quizzes; view grades of submitted documents and modeling of profiles which holds information such as user’s interests and characteristics of modules for personalization. A hybrid recommendation technique will be integrated for MEP. This technique will combine both content-based filtering and logic. This adaptive Web-Based system will use the modeling of documents and modeling of profiles which holds information such as user’s interests and characteristics of modules for personalization. MEP will make use of a hybrid recommendation technique by combining Lucene’s tag-based filtering, Lucene’s content-based filtering, adaptive sorting and fuzzy logic. MEP employs Lucene because it is a very good high-performance full text search engine. Fuzzy logic will determine the degree of truth in regards to the relevance of a link. MEP will employ this technique to enhance the available learning resources recommendation; hybrid recommendation technique will give better performance. Both information retrieval techniques will strengthen each other’s result. Students will have option of using either techniques or both, during a semantic search. Adaptive Navigation Support given to the student comes in form of linking to generated refined search solutions. Recommendation agent mediates and integrates all the knowledge sources. The agent platform sends recommendations to the content management who checks with the personalisation manager if the student request for any preferred method for displaying the result. Example, the student wants 5 results per page. The personalization manager is also in charge of promoting useful features that student made redundant. It offers student information of optional redundant features and offers options such as removal or trial. The agent platform, that searches for suitable modules, similar course mates, friends, resources and lecturers, works as a group of autonomous agents that cooperates with each other. They provide assistance on demand, at a time and location suitable to students’ needs. The agents will use high-level communication protocol (i.e. ACL messages) to update their findings. The content manager (this only occur when the search term employs content-based technique) and ontology model will invoke the agent platform which will pass search keywords to mediator agent. The mediator agent communicates with other agents, retrieve results which is then passed by the agent platform to the user interface. The user interface will display ranks of content, in accordance to similarity, user interest and academic strength on the user interface. Adaptive navigation support will enable students to manipulate the search results as the students navigate from one item to the other. This adaptive MEP use the modeling of documents and modeling of profiles which holds information such as user’s interests and characteristics of modules for personalization. A hybrid recommendation technique will be integrated for MEP. This technique will combine both content-based filtering and logic. This adaptive Web-Based system will use the modeling of documents and modeling of profiles which holds information such as user’s interests and characteristics of modules for personalization. MEP will make use of a hybrid recommendation technique by combining Lucene’s tag-based filtering, Lucene’s content-based filtering, adaptive sorting and fuzzy logic. MEP employs Lucene because it is a very good high-performance full text search engine. Fuzzy logic will determine the degree of truth in regards to the relevance of a link. MEP will employ this technique to enhance the available learning resources recommendation; hybrid recommendation technique will give better performance. Both information retrieval techniques will strengthen each other’s result. Students will have option of using either techniques or both, during a semantic search. Adaptive Navigation Support given to the student comes in form of linking to generated refined search solutions. Recommendation agent will collect search results and re-rank them in accordance to user’s interest and academic strength. Mediataor agent will broadcast the task; it will pass search results to Recommendation agent and...
pass users’ details to Data agent and Monitoring Agent. The Data and Monitoring Agents will collect information about the user from database and pass it to Recommendation agent. The Recommendation agent will now pass the final result to the Delivery agent. The Search agent is responsible for retrieving user’s details from the database.

3. CONCLUSION

The paper proposes MEP architecture for enhancing managed learning environment. MEP architecture is agent-oriented in which educational learning contents are enfolded by web services, the web services provides consistent services to students. This architecture is employed to provide recommendations and improve scalability, availability and personalization of the system. MEP aims to use enhanced wireless technologies, social networking, single supported MLE and context-awareness to boost the third level education for an intuitive and seamless learning process. A knowledge-based architecture will evaluate students’ interaction with the mobile platform for system retrieval, evaluation of its employment and filter of irrelevant redundant features.

4. REFERENCES

ABSTRACT
In this paper, we describe how multi-agent systems are used in teaching process to learn functionality of certain traffic control systems. Within introduction there is given a survey of those traffic models that had impact on consequently presented work. All the models in question are implemented in the hybrid compiler/interpreter NetLogo, other program environments are not considered. Then we emphasize the role of multiagent approach in a specific study program, its teaching concept and relation to provided courses. The main purpose in view is performance of traffic control systems through modeling their functional behavior – from the simplest to more complex ones. Descriptions of functionality are derived from fluid dynamics which helps to observe some required characteristics. The model of a railway level crossing operation serves to show how multiagent approach can be used to present basic functions of the traffic control system and observe characteristics such as the traffic moment, potentially employed in design of real equipment configuration. Some problems typical for newcomers to multiagent systems community are also presented together with description of the actual state and intents to near future.

Categories and Subject Descriptors
I.2.11 [Distributed Artificial Intelligence]: Multiagent Systems; K.3 [Computers and Education]: General.

General Terms
Performance

Keywords
Traffic, multiagent, level crossing, education, functionality

1. INTRODUCTION
A lot of work has been done so far in the field of traffic modeling based on the multiagent approach. The following survey is not exhaustive, however written with intention to cover the most decisive models from traffic domain that inspired the work presented later. Attention is paid to the NetLogo models applied in road transport only.

As the fundamental and introductory example the Traffic Basic model [10] can be considered, showing the movement of cars on a highway. Each car follows simple rules for deceleration or acceleration depending on presence of a car ahead. The model helps to demonstrate how traffic jams can form without any external (centralized) cause. More sophisticated two-lane version of this model can be found in [13], providing drivers a new option; they can react by changing lanes. In the model Traffic Intersection [12] cars are traveling through an intersection. The user has the ability to control the frequency of cars coming from each direction, the speed of the cars, and the timing of the light at the traffic intersection. Once the frequency and speed of cars is selected, the user should run the simulation and adjust the timing of the traffic light so as to minimize the amount of waiting time of cars traveling through the intersection. Another model (called Dangerous Drivers) demonstrates the flow of cars through a 4-way intersection with a traffic light. Each road approaching the intersection has three types of lanes: left-turn, right-turn, and straight [9]. More complex example can be found in the Traffic Grid model [11] where one may control traffic lights and overall variables, such as the speed limit and the number of cars, in a real-time traffic simulation, and try to develop strategies to improve traffic and to understand the different ways to measure the quality of traffic. An improved version of this model is available in [4] where traffic lights try to "self-organize" for improving traffic. Dresner and Stone in [3] propose a reservation-based system for alleviating traffic congestion, specifically at intersections, and under the assumption that the cars are controlled by agents, they show that their reservation-based system can perform two to three hundred times better than traffic lights. Another model [6] was inspired by a real-life incident at a 3-way T-shaped Indian intersection and shows how cars can be running under conditions of lights malfunctioning. Similarly, some models can be found dealing with pedestrian movement ([8], [2] and others).

2. TEACHING CONCEPT
2.1 Scope of the education process
The actually targeted group is represented by students of the 5th year of the MSc degree study program in Safety Control Engineering, who have previous knowledge on analysis and synthesis of safety-related control systems, dominantly applied in railway and road transport domains (ca 40 students graduating
The modeled control systems are mostly event driven systems, their functionality is described with the usage of UML, SysML, ladder-logic (for PLCs) or other formalisms; technical safety is analyzed in BQR, CARMS or other reliability engineering software.

The academic year 2008/2009 is the last year of the 5 year transition period during which the 2-stage study program has been transformed to the 3-stage one. This change will bring strengthening position of AI in the education process and better integration of multiagent systems since as a different modeling approach that makes possible to seat the system into certain environment and observe how its behavior is evolving over time. What’s more, AI-related courses will also become provided to more study programs at the faculty. At present, multiagent systems represent a marginal topic supplementing knowledge representation, uncertainty processing, expert systems, machine learning and other topics. Essential theory background is complemented with some minimum practical experience based on work with the NetLogo environment. Students are motivated to master their models from simple to more complex ones; however number of lessons and exercises is insufficient.

3. Teaching Concept

The presented teaching concept is based partially on the first obtained experience and partially on intentions associated with the mentioned transformation of study programs. The teaching process is and will be realized in the consecutive steps.

3.1.1 Motivation

At first students gain a basic understanding of the NetLogo environment and are introduced to agent behavior and emergent properties. To motivate them some working model is explored to show how agent modeling is used and what NetLogo capabilities are. An existing model can be slightly modified to see how different pieces affect the functionality of the model and how the software environment can be handled. Students familiarize themselves with the NetLogo interface – how to use tabs, buttons, sliders, switches, monitors and plots, and what are patches, turtles and observer.

3.1.2 Fundamentals Encompassment – Patches

In the next step a simple model is built piece-by-piece using a pseudo-code approach which involves writing behaviors in comments before using the code. At first attention is paid to patches and creating the simulation world. A series of subtasks aims to create a surface communication, beginning with one straight lane through intersections to multilane and/or complex trajectories (see Figure 1).

3.1.3 Fundamentals Encompassment – Turtles

Having basic knowledge on usage of patches, next attention is paid to turtles, i.e. creating cars from drawing their shapes to principles of their generating, killing, re-directing, moving and changing properties.

3.1.4 Creating Scientific Models

Gained knowledge then can be used to create scientific models, i.e. models used for testing hypotheses, analyzing scenarios, drawing conclusions to certain problems etc. In relation to road traffic the following car movement strategies can be realized:

a) Cars are moving with regular spacing at constant velocity;
b) Cars react to presence of another car being ahead (rules of decelerating and accelerating), and/or other side-ways effects (presence of police patrol, a traffic sign ordering speed restriction, etc.);
c) Cars are able to change lanes;
d) Cars follow certain traffic rules when passing intersections;
e) Cars follow rules given by a certain traffic control system (traffic light controller, level crossing signaling; platoons control in automates highway systems etc.).

The last item represents the target group of models that makes possible to solve different problems of optimization, control strategies, traffic operation under malfunctioning conditions and so on, inevitably based on observing certain parameters of traffic flows.

3.1.5 Self-activity of Students

While all the students should absorb basic knowledge of multiagent systems, only a few of them still have a chance to solve complex problems mentioned in the previous paragraph. At present self-activity is performed mostly in the form of MSc thesis or term projects. In near future more time should be reserved and spent over topics in question.

3.2 Railway Level Crossing Example

To demonstrate the first outcomes of the described teaching concept the model of a railway level crossing example is briefly presented. The very first idea of making the model appeared in 2005 when a concept model was designed and submitted to the NetLogo community.¹ Then it was used as a motivation example in the AI-related courses. It suffered from many simplifications

![Figure 1. Modeling of patches to create a world.](http://ccl.northwestern.edu/netlogo/models/community/LevelCrossing)
but became valuable for acquisition of necessary skills and for getting the modelling environment under control. In the academic year 2006-2007 the model was used as a starting point for elaboration of the MSc thesis. The new updated model was made more realistic and helped to move attention from “how to do something” to “what/why to do so”. One year later the model was further improved, translated from Slovak to English and presented to the NetLogo community. Technical and implementation details of the model addressing community of intelligent transport system professionals were presented in [6].

Control elements applied in the NetLogo interface (see Figure 2) make possible to set parameters such as road vehicle types, the way of their arrivals and maximum speed (separately for each direction), configuration of freight and passenger trains, selection of train graphic timetable, maximum railway line speed, possibility to generate an auxiliary train, usage of approaching time predictor. This makes students consider and understand causalities and background theory as used in real life. For example, if constant approach time for different speeds of rail vehicles is to be fulfilled, we need variable length of the approach section. In the real life this length is defined as:

\[
L_p = \frac{28 \cdot t \cdot v_t}{s} \text{ [m; km·h}^{-1}, \text{s]} ,
\]

where \(L_p\) is length of the approach section and \(v_t\) is max allowed velocity at the rail line. The variable \(t\) is time decisive for calculation of length of the approach section and in reality it is composed of more partial times (approaching time, human reaction time, reaction time of equipment, time of barriers going down, additional safety time, etc.). Not all partial times are considered in the model.

One of the outcomes of the model is the static parameter \(M\) called “traffic moment” that can be obtained at the end of one simulation day. This parameter is important when sufficiency or insufficiency of technical equipment installed at the level crossing is evaluated. According to valid legislation traffic moment \(M\) is officially defined as a product of two traffic densities

\[
M = D_{\text{road}} \times D_{\text{rail}}
\]

where \(D_{\text{road}}\) is road traffic density and \(D_{\text{rail}}\) is railway traffic density. Calculation of the traffic moment in the model is slightly simplified (e.g. no pedestrians and bicyclists are considered), but its principal idea can be demonstrated and presented to students. In the real life the value of traffic moment for level crossings installed at the main lines is usually about \(5 \cdot 10^5\). If the simulation of the model is terminated in a standard way, statistic data is automatically exported from the model and saved to a separate file for the next analysis.

---

2 Janota, A. 2008. LevelCrossing ver2_1
http://ccl.northwestern.edu/netlogo/models/community/LevelCrossing%20ver%202_1
3.3 Existing Problems

Wider use of multiagent systems in the educational process assumes overcoming several essential problems:

a) Insufficient time and space reserved in the education process: the situation should improve in the coming academic year after completing transition from the old study system to the new one; the basic knowledge will be given in the course called Expert Systems which will also be offered to more study programs at the faculty; more implementation details and practical work will be provided in the course called AI Programming; self-activity oriented to advanced multiagent applications will be supported in other courses emphasizing work over own projects (Term Project, Diploma Project), and/or some MSc theses;

b) Lack of study literature concerning the NetLogo environment: at present two primary sources of information are used – the help documents built-in in the NetLogo environment and community models available from the Internet; however English speaking abilities of the students are often low and for near future some Slovak textbook on NetLogo could be prepared;

c) Low number of people involved in teaching multiagent systems at the faculty/department and absence of closer contacts and cooperation with the multiagent community; therefore this paper was written with motivation to inform about the actual situation and authors’ intentions.

4. CONCLUSIONS

The paper was written with motivation to show how multiagent systems can be used as a supplementary approach to simulation and study of complex systems. The level crossing model is presented to demonstrate a typical output of realized teaching concept as described above, with a minimum theory hidden in the background.

5. ACKNOWLEDGMENTS

The paper was prepared under support of the Slovak Grant agency VEGA, grant No. 1/0023/08 “Theoretical apparatus for risk analysis and risk evaluation of transport telematic systems”.

6. REFERENCES


