MAS Coursework Design in NetLogo

3 authors:

Ilias Sakellariou
University of Macedonia
54 PUBLICATIONS **187** CITATIONS
SEE PROFILE

Petros Kefalas
The University of Sheffield
122 PUBLICATIONS **785** CITATIONS
SEE PROFILE

Ioanna Stamatopoulou
The University of Sheffield
49 PUBLICATIONS **244** CITATIONS
SEE PROFILE

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,

Cite as: MAS Coursework Design in NetLogo, Ilias Sakellariou, Petros Kefalas and Ioanna Stamatopoulou, *EDUMAS 09 Workshop of the 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, M. Beer, M. Fasli and D. Richards (eds.), May, 12, 2009, Budapest, Hungary, pp. 47-54.

- 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 most 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,

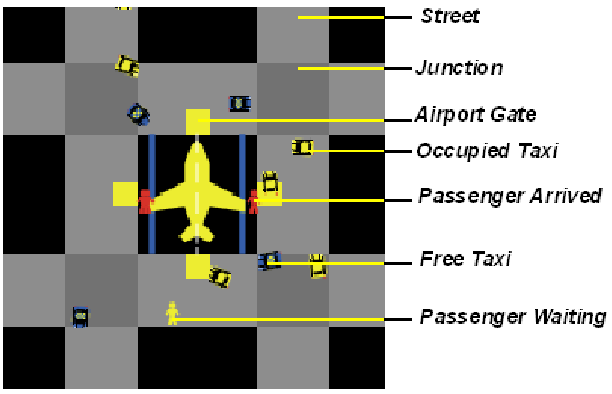


Figure 1: Entities in the scenario environment

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:

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

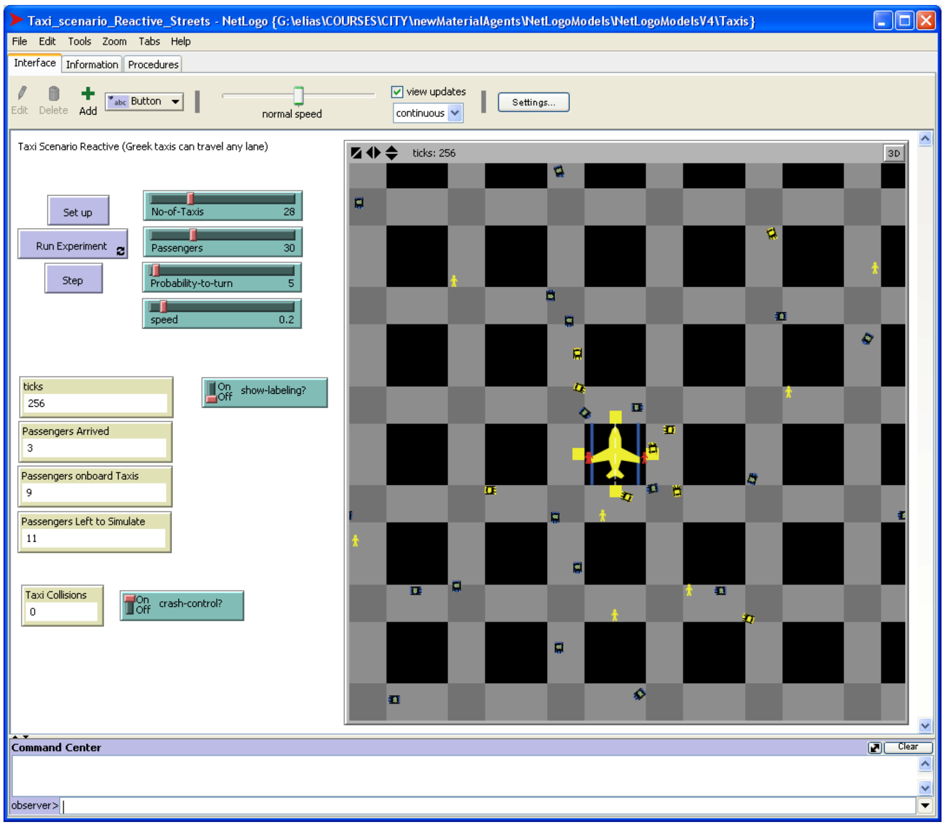
Note how easily such a sensor can be implemented, given the NetLogo primitives and the environment representation described in section 3.1. A set of agent actions was also released with the handout. The code that follows, shows the implementation of three taxi agent actions, the first one stochastically 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.

```
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.


Taxi_scenario_Reactive_Streets - NetLogo {G:\elias\COURSES\CITY\newMaterialAgents\NetLogoModels\NetLogoModelsV4\Taxis}

Taxi Scenario Reactive (Greek taxis can travel any lane)

No-of-Taxis 28

Passengers 30

Probability-to-turn 5

speed 0.2

ticks 256

Passengers Arrived 3

Passengers onboard Taxis 9

Passengers Left to Simulate 11

Taxi Collisions 0

show-labeling? Off

crash-control? Off

view updates ✓ continuous

normal speed

ticks: 256

Figure 2: The scenario environment in NetLogo

```
to execute-taxi-behaviour
   if not have-passenger and
      detect-passenger [pick-up-a-passenger stop]
   if detect-taxi [turn-away stop]
   if detect-street-edge-left [rt 5 stop]
   if detect-street-edge-right [lt 5 stop]
   if have-passenger and
      reached-airport [drop-passenger stop]
   if have-passenger [move-to-airport stop]
   if detect-junction [move-randomly-90 stop]
   if true [move-ahead stop]
end
```

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.

Manipulating Beliefs
`create-belief [b-type content]`
`belief-type [bel]`
`belief-content [bel]`
`add-belief [bel]`
`remove-belief [bel]`
`exists-belief [bel]`
`exist-beleifs-of-type [b-type]`
`beliefs-of-type [b-type]`
`get-belief [b-type]`
`read-first-belief-of-type [b-type]`
`update-belief [bel]`
Manipulating Intentions
`execute-intentions`
`get-intention`
`intention-name [intention]`
`intention-done [intention]`
`remove-intention [intention]`
`add-intention [name done]`
Message Exchange
`create-message [performative]`
`create-reply [performative msg]`
`add-sender [sender msg]`
`add-receiver [receiver msg]`
`add-multiple-receivers [receivers msg]`
`add-content [content msg]`
`to-report add [msg field value]`
`get-performative [msg]`
`get-sender [msg]`
`get-content [msg]`
`get-receivers [msg]`
`send [msg]`
`receive [msg]`
`get-message`
`remove-msg`
`broadcast-to [breed msg]`

Table 1: Primitives provided to students of the FIPA ACL Message Passing and BDI NetLogo libraries

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:

```
;;; Plan to find a taxi (in reverse order)
to 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
```

Note that the add-intention call adds an intention to a stack, and that the intention persists until its condition becomes true. The first argument of the 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:

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

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:

```
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
end
```

The reactive layer consists of three rules that implement collision avoidance. If any of the rules fires, control does not proceed to the 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.

```
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] ]
end
```

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 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 plan-to-pickup-passenger procedure:

```
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 "drop-passenger" "true"
 add-intention
   "carry-passenger-to-airport" "reached-airport"
 add-intention
   (word "check-passenger-onboard" pass_no)
     "true"
 add-intention
   (word "pick-up-passenger " pass_no) "true"
 add-intention (word "move-to-dest " coords)
   (word "at-dest " coords)
 add-intention (word "move-to-dest" junction)
   (word "at-dest " junction)
end
```

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 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:

```
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"
  ]
end
```

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

[1] M. D. Beer and R. Hill. Teaching multi-agent systems in a UK new university. In *Proceedings of 1st AAMAS Workshop on Teaching Multi-AgentSystems*, 2004.

[2] M. D. Beer and R. Hill. Multi-agent systems and the wider artificial intelligence computing curriculum. In *Proceedings of the 1st UK Workshop on Artificial Intelligence in Education*, 2005.

[3] R. H. Bordini. A recent experience in teaching multi-agent systems using Jason. In *Proceedings of the 2nd AAMAS Workshop on Teaching Multi-Agent Systems*, 2005.

[4] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[5] M. Fasli and M. Michalakopoulos. Designing and implementing e-market games. In *Proceedings of the IEEE Symposium on Computational Intelligence in Games*, pages 44–50. IEEE Press, 2005.

[6] Foundation for Intelligent Physical Agents. *FIPA ACL Message Structure Specification*, 2002. www.fipa.org/specs/fipa00061/.

[7] Foundation for Intelligent Physical Agents. *FIPA Contract Net Interaction Protocol Specification*, 2002. www.fipa.org/specs/fipa00029/.

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

and planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 677–682, 1987.

[9] H. Hara, K. Sugawara, and T. Kinoshita. Design of TAF for training agent-based framework. In *Proceedings of 1st AAMAS Workshop on Teaching Multi-AgentSystems*, 2004.

[10] P. Kefalas, I. Stamatopoulou, I. Sakellariou, and G. Eleftherakis. Transforming Communicating X-machines into P Systems. *Natural Computing*, 2008. To appear.

[11] I. Sakellariou. Extending NetLogo to support BDI-like architecture and FIPA ACL-like message passing: Libraries' manuals and examples. http://eos.uom.gr/∽iliass/projects/NetLogo.

[12] I. Sakellariou, P. Kefalas, and I. Stamatopoulou. Enhancing NetLogo to simulate BDI communicating agents. In *Artificial Intelligence: Theories, Models and Applications, Proceedings of the 5th Hellenic Conference on AI (SETN'08)*, volume 5138 of *Lecture Notes in Computer Science*, pages 263–275. Springer, 2008.

[13] I. Sakellariou, P. Kefalas, and I. Stamatopoulou. Teaching intelligent agents using NetLogo. In A. Cortesi and F. Luccio, editors, *Proceedings of Informatics Education Europe III (IEE-III)*, pages 209–221, 2008.

[14] U. Wilensky. Netlogo. Center for Connected Learning and Computer-based Modelling. Northwestern University, Evanston, IL. http://ccl.northwestern.edu/netlogo., 1999.