# SET10117 Multi-Agent Tutorial Timetable Generation

Calum Hamilton

40205163

### Introduction:

In this coursework, I was assigned the task of prototyping a decentralised multi-agent system which would assign timetable slots to students based on their preferences. Timetable slots are initially assigned to student agents (acting on behalf of a student) who attempt to swap their slot until they receive preferential times.

### Design:

My program design and report will address the following five design requirements:

1. A meaningful and relevant ontology, allowing agents to communicate
2. A communication protocol, allowing agents to communicate without revealing their preferences
3. A utility function, allowing the student to know when they’re satisfied with their slots
4. A strategy to determine which exchange requests to make, which to accept and which to reject
5. A metric to evaluate system effectiveness in terms of satisfying student preferences

#### 1. Ontology Design:

I include my Ontological Relationships in Appendix A. and examine the Ontological elements here.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Element** | **Properties** | **Data Type** | **Restrictions** | **Type** |
| **Pleased With** | Student Identifier | AID | Mandatory | Predicate |
| **Timeslot** | Day  Time | Integer  Integer | Mandatory  Mandatory | Concept |
| **Tutorial Group** | **Timeslot**  Tutorial ID  Class Size  Tutorial Number | Ontology Element  String  Integer  Integer | Mandatory  Mandatory  Mandatory | Concept |
| **Swap Initial** | **Tutorial Group**  Agent From | Ontology Element  AID | Mandatory  Mandatory | Agent Action |
| **Swap Final** | **Swap Initial**  **Tutorial To**  Agent To | Ontology Element  Ontology Element  AID | Mandatory  Mandatory  Mandatory | Agent Action |
| **Unhappy Slot** | **Swap Initial** | Ontology Element |  | Agent Action |
| **Slots Requested** | ArrayList<**Swap Final**> | List<Ontology Elements> | Mandatory | Predicate |
| **Happy With** | ArrayList<**Swap Final**> | List<Ontology Elements> | Mandatory | Agent Action |
| **Message Board** | ArrayList<**Swap Initial**> | List<Ontology Elements> |  | Concept |
| **Slots Available** | **Message Board** | Ontology Element | Mandatory | Predicate |

#### 2. Communication Protocol:

There were several ways I envisaged the communication protocol for my multi-agent system:

* A free-for-all messaging system
* Students sending unwanted slots to an agent that broadcasted the slots to students
* Having a message board where slots could be posted and requested from

My biggest focus when planning the communication protocols, was allowing the most communication in the fewest messages. This included live messages and total messages.

My foremost concern with a free-for-all messaging system is that it would pass too many live messages– meaning that messages may not be received. Additionally, I was concerned it would create too many messages overall – especially if you the solution is scaled. It may be reasonable for 10 agents to send and request slots at the same time, but the number of messages sent is nn per iteration, where n is the number of agents. Plus, anyone who has been at a large Christmas gathering can tell you that if everyone is speaking to everyone else at the same time, it leads to inefficient communication.

An Agent receiving and broadcasting slots to Student Agents would help limit the number of messages that a free-for-all messaging system would cause. Allowing for one slot per iteration to be offered, the complexity of this is still 2n per timestep, where n is the number of agents (one message to the broadcaster from the sender, one message sent to other agents, one reply from each of the agents and one reply from the broadcaster). You could attempt to limit the messages further by having agents not respond if they’re not interested in the slot, but it still runs into a lot of the issues that the free-for-all messaging system would run into.

An underlying reason for these issues is because Student Agents aren’t allowed to expose their timeslot preferences (e.g. can only respond yes/no). Since Agents can’t expose their preferences, the receiver is unable to know which agents would be best benefitted, given a certain slot (can’t choose the result that maximises social welfare).

This led me to understand the best approach is to give a Student Agent a selection of slots and let them choose the slot best suited to them, as opposed to passing single slots and having them return yes or no. This also points towards a centralised approach – having the ability to store and pass multiple slots simultaneously.

I realised that the Communication Protocol would require a controller agent to limit the number of live messages and store a bank or ‘message board’ of unwanted slots. I started using a ‘Timetabling Agent’, to hold a list of slots for agents to swap.

I detail the Communication Protocol design in the Sequence Diagram in Appendix B.

#### 3. Utility Function:

There were several ways that I saw the Agent utility being calculated:

* Individual calculation
* Mean utility
* Specialised calculation

You could calculate the individual utilities of each module that an Agent is assigned and use the individual calculated value to choose whether the Agent is ‘happy’ or needs to swap a slot by iterating through the individual values. This is suitable enough for selecting which slots to swap or comparing an existing slot with another slot. When it comes to evaluating the ‘happiness’ of an Agent with >1 slot it becomes more difficult. This is where you need an aggregated calculation.

Using a mean utility function, you could compare slots individually for comparing slots to post or comparing a swap request. However, it allows you could calculate a mean happiness based on the Agent’s slots. Using mean utility as the Agent utility function appealed to me, as it makes sense that the ‘happiness’ of the Agent is based on the average slot happiness. From a human-based perspective:  
“I really hate that 5pm Friday tutorial, but my other slots are nice, so overall my timetable isn’t too bad”.

A specialised calculation, or an adaption of the mean utility, could be a good choice given the task. It’s difficult to tell how this would be shaped ahead of time, but it would most likely include additional dissatisfaction if an Agent couldn’t attend a slot.

##### Choice of Calculation:

I saw the choice as either: using the number (and by extension the mean of the number) or using an inverse. For me, it made more sense using an inverted range of numbers to provide the Agent preferences. My slot preferences range from 0-4:

1. Agent loves this slot (high preference)
2. Agent likes this slot
3. Agent is indifferent about this slot (medium preference)
4. Agent doesn’t like this slot
5. Agent can’t attend this slot (low preference)

It made sense using a range of five numbers instead of three, to capture the range of human emotion more fully towards slots (liking/indifference/hating seemed too restrictive), without being too complex.

#### 4. Strategy for Considering Requests and Exchanges to Make

In the system I designed, there were three opportunities for students to swap slots:

1. Informing the Timetable Agent of slots they were interested in
2. Agreeing to/Refusing slots that other agents had requested
3. Posting new slots on the message board they didn’t want

I examine these more deeply below:

1. The Student Agent needed to evaluate its slots against slots on the message board. The student would then be able to select good slots/the best slot from and request those slots.  
   I limited the number of unique tutorials slots the student could request to one tutorial per Set ID. I found this a good balance between requesting too few slots each cycle and slowing down the system with a list of requests to be verified one-by-one. If an Agent could request multiple tutorials with the same Set ID, the Timetable Agent would need to verify if the first Agent was happy to swap, then the second if the first wasn’t happy etc.
2. I had Student Agents refuse swaps if the new slot fitness was worse or equal to their current. They would only accept a new slot if it was better than or equal to their current slot. This meant that Agents would post slots they didn’t like but wouldn’t swap them if the one offered was worse.
3. I originally had agents posting their worst slot and then had the Timetable Agent decline it if the slot was already present on the message board. This however meant that not enough slots were being posted to swap quickly and efficiently, and that Agents would still be stuck with slots they didn’t want. I updated this by having the Agents post slots they were unhappy with, that they hadn’t already posted – alternately post no slots.

#### 5. A Metric to Evaluate Overall Effectiveness of the System

I thought the best way to evaluate the system was a somewhat Utilitarianist approach in trying to understand the happiness of Agents across the system. I evaluate the ‘happiness’ of each Actor (Actor fitness) through a custom evaluation function, and this is what I want the system to maximise.

Since this is what I want to maximise, it seemed appropriate to average the custom evaluation function across Agents and get a Mean. The metric I’m using to evaluate overall system effectiveness is the mean value of the Utility Function – I’m calling Mean System Fitness. I examine this Mean System Fitness in each of my test cases I present below.

### Implementation

My Implementation Report will address the following three requirements:

1. Show the utility calculation, where it decides which swap requests to accept and which to reject, and state where the student’s preferences are represented
2. State where in the code the timetabling agent ensures that each student attends exactly one tutorial for each module.
3. For each conversation in your communication protocol, you should reference the relevant screenshot from the JADE sniffer, and state which agent behaviours implement it.

#### 1. Student Utility Calculation and Preference Exchange:

I originally planned to implement a mean calculation. The mean utility function worked very well on my first 3 test cases (less users and modules). After testing the mean utility on my fourth test case, I realised I was still accepting Student Agents if they had a slot but couldn’t attend. I tried to fix this by raising and dropping the fixed threshold for ‘happy students’, but this didn’t solve the issue. I decided that the mean calculation was unfit for purpose. I adapted a specialised calculation from the original mean value calculation I was using. The formula I settled on was:

Where *p* = Slot Preference (ranging 0 – 5) – 0 means the Student Agent loves the slot; 5 means the student can’t attend that slot. -5 is a low enough value where the student can never accept the slots without swapping all slots they can’t attend. Slots the student loves and can’t attend are heavily weighted. This calculation is shown in Appendix C.

Output values (for a single example):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Slot Pref:** | 0 - loves the slot | 1 - likes the slot | 2 - indifferent | 3 - doesn't like | 4 - can't attend |
| **Value:** | 1.0 | 0.5 | 0.33° | 0.25 | -5 |

I found if I started the utility function’s target happiness at a reachable near-optimal level (e.g. 0.7-0.9), Student Agents would swap several slots and then exit, without considering any other slots available. This left less Student Agents in the environment, with less opportunity for Student Agents to swap bad slots. I tested various happiness thresholds and found that 1.5 with a reduction of 0.02 upon each time they were notified to request slots was a good level. It didn’t run for too long and allowed Student Agents that had selected optimal slots to exit the environment early.

##### Preference Representation & Swap Requests:

I examine the three areas above, in design section 4. For clarity, I will re-explain the areas of calculation below:

1. Informing the Timetable Agent of slots they were interested in
2. Agreeing to/Refusing slots that other agents had requested
3. Posting new slots on the message board they didn’t want

Appendix D shows how the Student Agent selects ‘interested slots’ – the individual best slots to request for each tutorial class. This is in the **AwaitTimetableBehaviour** behaviour in the Student Agent class.

Appendix E shows how the Student Agent selects which slots to agree to swap and which to refuse. This is in the **AwaitSlotVerifyBehaviour** behaviour in the Student Agent class.

Appendix F shows how the Student Agent selects which new slots to post to the message board. This is in the **AwaitHappyBehaviour** behaviour in the Student Agent class.

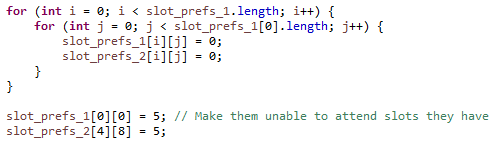
I examine the example conversations in section 3 of the implementation, but the screenshots of the test case 1 conversation are shown in Appendix G.

The Student Agent slot preferences are generated in a 2D array:



Corresponding to slot\_preferences[days\_in\_week][hours\_in\_day].

Below is how the slot preferences have been generated for the Test Case 1:



#### 2. Ensuring Unique Module Attendance:

When the program begins, Student Agents are each assigned their week’s slot preferences, and the tutorial slots which they will be attending. The Timetable Agent is also passed the tutorial slots (but not the slot preferences).

The timetable then holds a 2D array of ArrayLists, which is checked whenever a Student Agent makes a slot request and updated whenever two Student Agents make a slot swap. The StudentSlots structure is in the format: - StudentSlots[day][time]. It uses references to the TutorialMap – a HashMap with the Tutorial ID as keys and a corresponding number, referring to the slot ordering.

Appendix H shows the Timetable Agent checking:

* The SETID of the slot requested is the same as the SET ID of the slot being given
* The Student Agent owns the slot it’s trying to swap

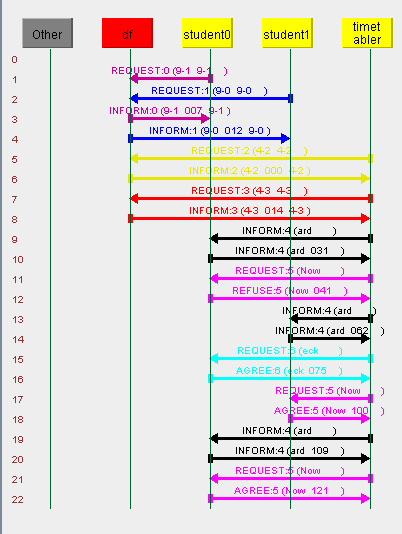
Appendix I shows the Timetable Agent updating the slots for the Requester and Poster Agents.

Appendix J shows the Timetable Agent checking to ensure that the Poster Agent owns the posted slot.

This approach can be guaranteed to work, as when Student Agents are being assigned modules, they will only ever be assigned a module with 1 SET ID. The Timetable Agent facilitates swaps directly from one Student Agent to another, ensuring to check the slots belong to the same slots and that the Student Agents own the slots.

#### 3. Exploring Agent and Timetable Agent Conversations

In this section, I explore the messages exchanged between two Student Agents and the Timetable Agent in a simple environment. Below is a screenshot of messages exchanged, taken through the Sniffer agent:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Line(s)** | **Agent from** | **Agent to** | **Purpose** | **Code Function** |
| 0-6 | All | DF | System registration | Register |
| 7-8 | Timetable | DF | Requesting Student list | FindStudentsBehaviour |
| 9 | Timetable | Student0 | Informing of timetable slots (0 slots total) | SwapStudentBehaviour |
| 10 | Student0 | Timetable | Informing of interested slots | AwaitTimetableBehaviour |
| 11 | Timetable | Student0 | Asking if unhappy or has slots | SwapStudentBehaviour |
| 12 | Student0 | Timetable | Unhappy – responds with slot | AwaitHappyBehaviour |
| 13 | Timetable | Student1 | Informing of timetable slots (1 slot total) | SwapStudentBehaviour |
| 14 | Student1 | Timetable | Informing of interested slots (Student0’s slot) | AwaitTimetableBehaviour |
| 15 | Timetable | Student0 | Ask if Student0 wishes to swap its slot for Student1’s slot | SwapStudentBehaviour |
| 16 | Student0 | Timetable | Agrees to swap for Student1’s slot | AwaitSlotVerifyBehaviour |
| 17 | Timetable | Student1 | Notifies of agreed swap and asks if happy | SwapStudentBehaviour |
| 18 | Student1 | Timetable | Happy – responds and terminates | AwaitHappyBehaviour |
| 19 | Timetable | Student0 | Informing of timetable slots (0 slots total) | SwapStudentBehaviour |
| 20 | Student0 | Timetable | Informing of interested slots | AwaitTimetableBehaviour |
| 21 | Timetable | Student0 | Asking if unhappy or has slots | SwapStudentBehaviour |
| 22 | Student0 | Timetable | Happy – responds and terminates | AwaitHappyBehaviour |

Screenshots of the conversation from lines 9-22 are captured in order in Appendix G. This implementation is an accurate implementation of the original system design.

### Testing

***Describe your test cases, and justify why you have chosen them. Present the results from running your system on each test case, and justify the output metrics that you have chosen***

***AND SHOW MEAN SYSTEM FITNESS***

**TOP MARK:**

At least three test cases are chosen where their justification demonstrates an excellent understanding of the problem. The results are clearly presented with appropriate and clearly explained output metrics. The test cases clearly demonstrate how the system performs with increasing difficulty of the problem.

### Evaluation and Future Work:

1. How will the effectiveness of your system in will change as the problem becomes more difficult?
2. What are the advantages and disadvantages of taking a multi-agent systems approach to this problem?
3. In light of 1 and 2, suggest and justify an improvement to your system.

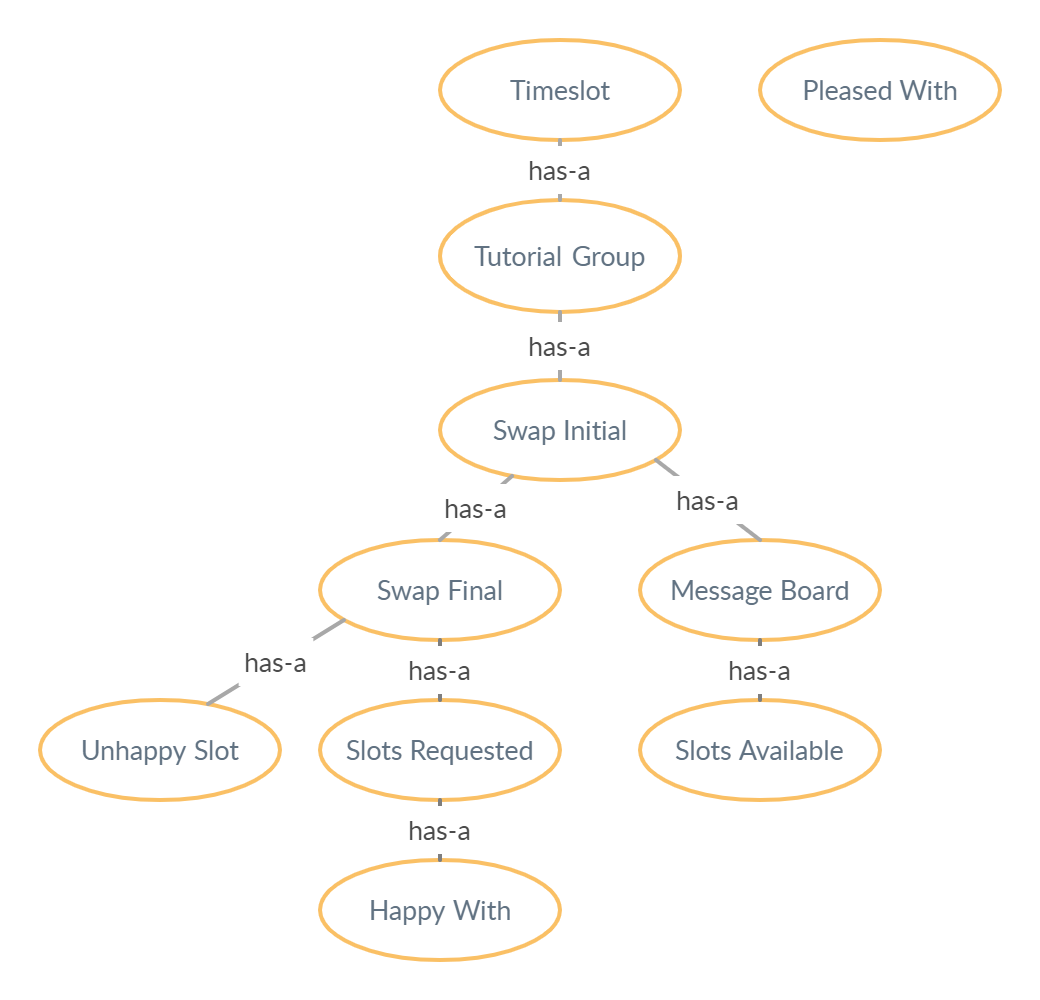
#### Future Work

* Have number of students allowed in each class and have the timetable agent post slots
* Have the timetable agent not send the message board if the message board is empty and immediately go to step 5

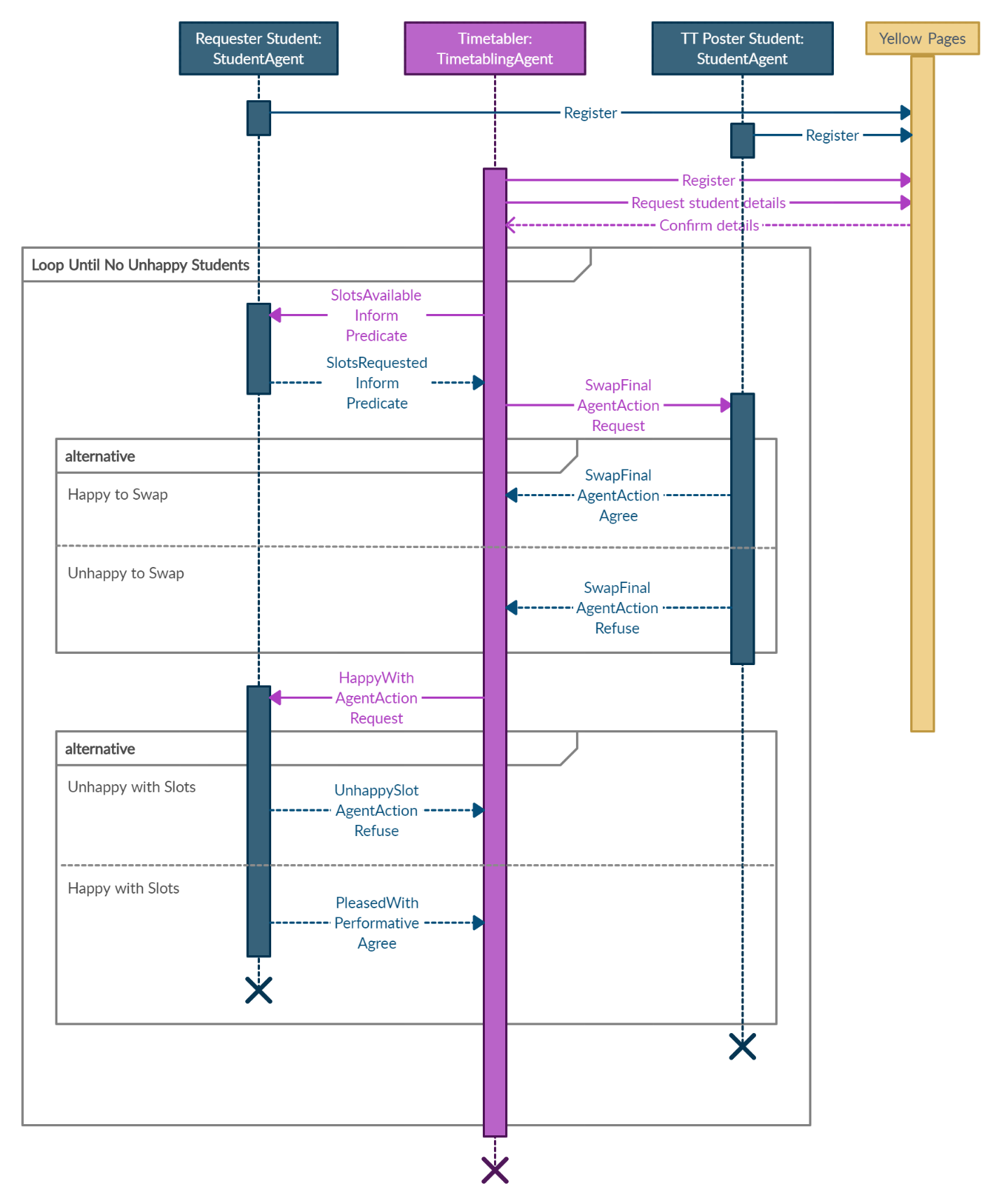
### References

### Appendix:

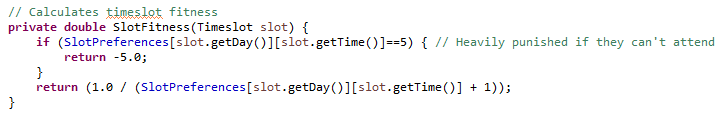
#### Appendix A – Ontology Association



#### Appendix B – Message Sequence Diagram



#### Appendix C – Student Agent Preference Calculation



#### Appendix D – Selecting and Requesting ‘Interested Slots’ from the Timetable Agent



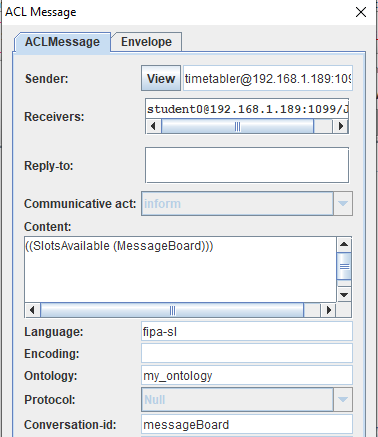
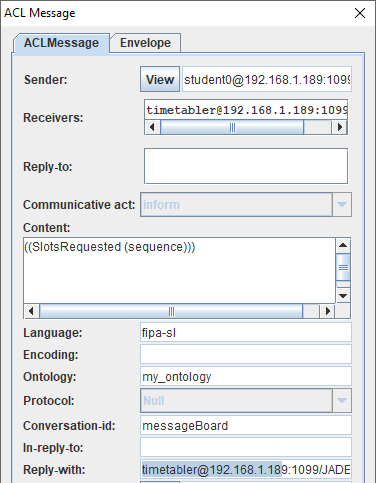
#### Appendix E – Agreeing to or Refusing ‘Requested Slots’ from the Timetable Agent

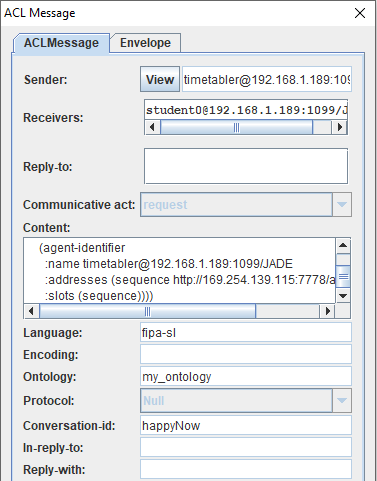
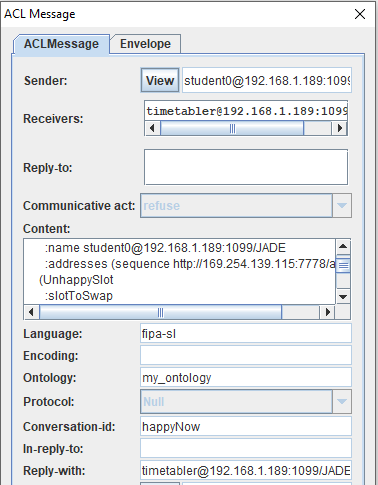


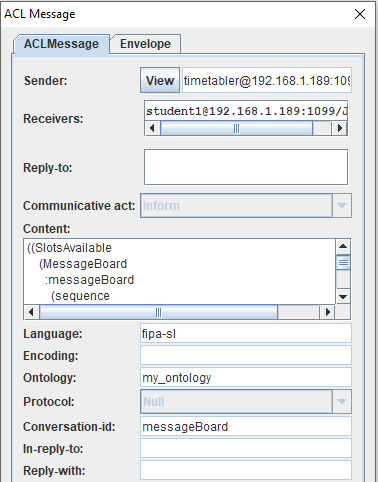
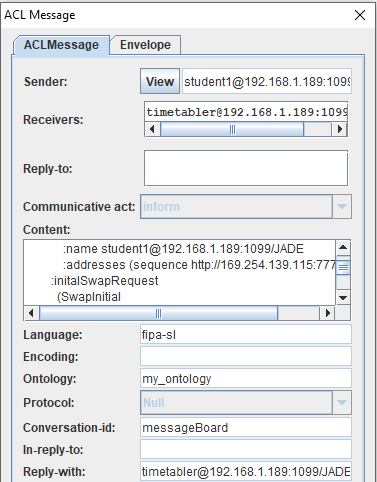
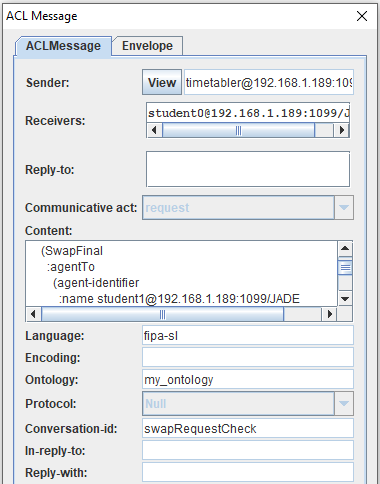
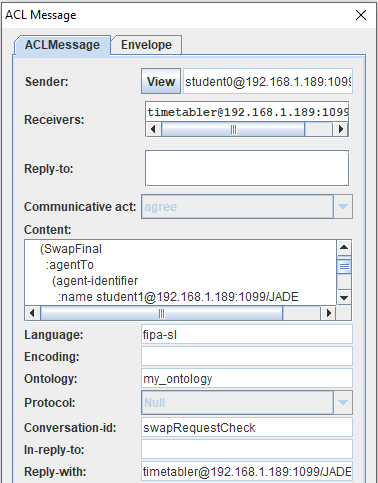
#### Appendix F – Checking new Slots to Post

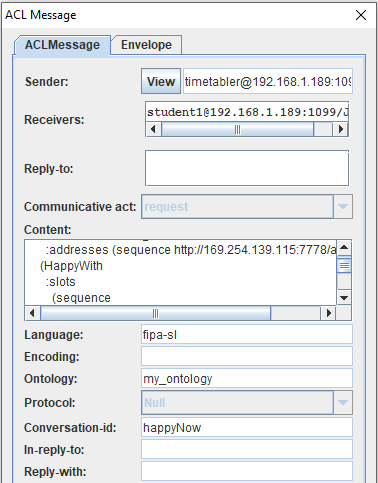
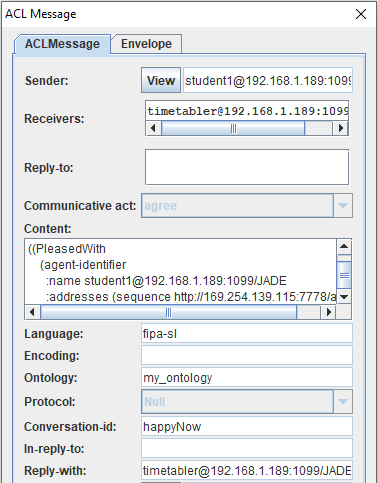
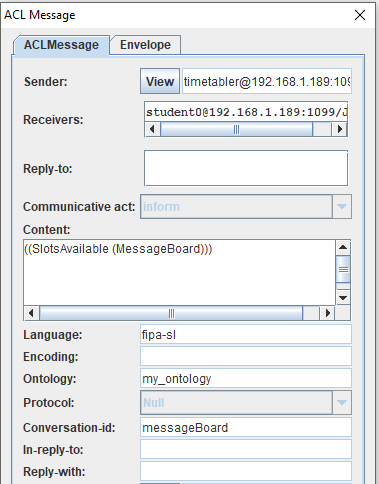
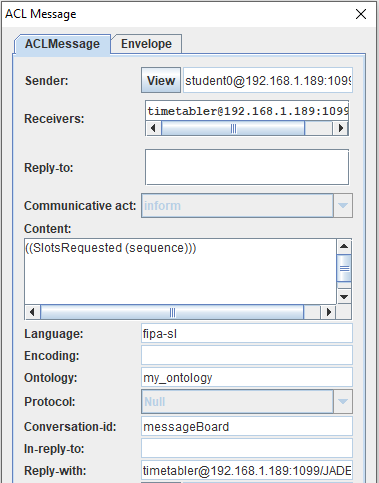


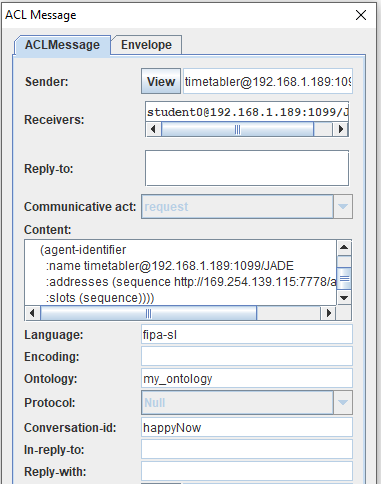
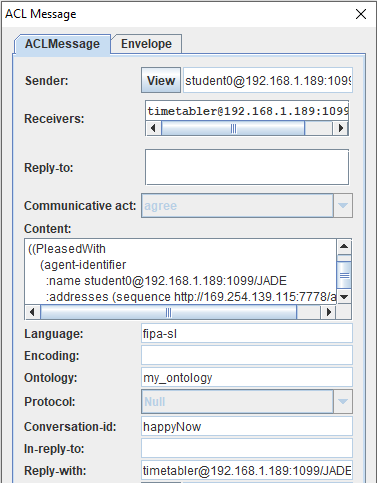
#### Appendix G – Example Conversations for Initial Example

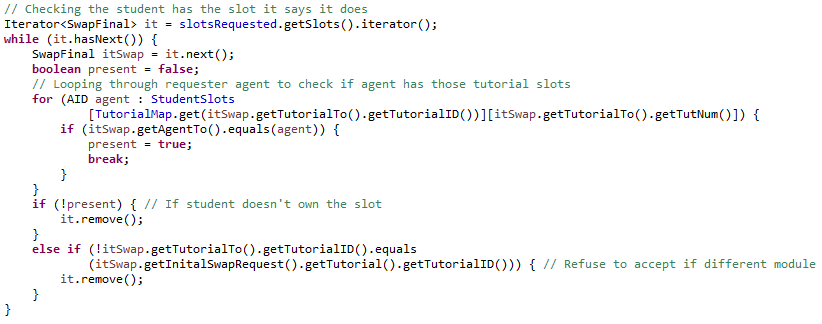
 

#### Appendix H – Denying Slot Request if Invalid



#### Appendix I – Updating the Requester Agent and Poster Agent’s slots



#### Appendix J – Checking the Poster Agent Owns the Posted Slot