Multi-Agent Systems
SET10111 Multi-Agent Systems Coursework

Aaron Campbell

Word Count: 3617

Table of Contents

Chapter 1 – Design

1.1. Ontology

In this system, there is an ontology that is representative of a student's timetable. Inside a Timetable there are Tutorials, which can be represented as a Concept. These are representative of the actual tutorials inside the timetable e.g., Multi-Agent Systems inside a 4th year Computing Science student's Timetable. There are Timeslots, which are representative of Timeslot objects in the Timetable i.e., a tutorial can be run on a Tuesday and a Friday – Tuesday and Friday's tutorials are both individual Timeslots - these are Predicates. There must also be Available Timeslots which are Predicates and representative of Timeslots that have been marked as potentially swappable by Student Agents, as in been put up for a potential swap by the Timetable Agent from a message. Finally, there are Swap Proposals inside the ontology, which are representative of proposed swaps in the system – these are AgentActions as they represent an actual action carried out by the agent.

To specify the ontology properly, it must be arranged into its hierarchy. In this case:

- A Timeslot is an instance of a Tutorial
- Available Swaps are instances of Timeslots
- Swap Proposal is an action which 'has a' Timeslot.

Ontologies must also be described in terms of the properties inside the concepts in the domain. In this system, a high-level description is as follows:

- Tutorial
    - Attendee - represented as an AID
    - Module Name – represented as a String
    - Module Number – represented as a String
    - Campus – represented as a String
    - Lecturer – represented as a String
    - Day – represented as a String
    - Start Time – represented as an Int
    - End Time – represented as an Int

- Timeslot
    - Timeslot Attendee – represented as an AID
    - Tutorial – a Tutorial, as in the Tutorial this Timeslots belongs to.

- Available Timeslots
    - Timeslots – an ArrayList of Timeslots, which can be accessed through its Getter and Setter

- Swap Proposal
    - Timeslot Owner/Attendee – represented as an AID, the Student holding the Timeslot.
    - Timeslot Receiver – represented as an AID, the Student receiving the Timeslot.
    - Timeslot – a Timeslot, the actual Timeslot is swapped.

1.2. Communication Protocol

Urquhart & Powers (2019) identify two types of agent in a multi-agent system: a Problem Agent and Stakeholder Agents. Problem agents are responsible for maintaining and ensuring the validity of the system. Stakeholder agents represent the interests of each individual 'stakeholder' in the system. In this system, the basic agents which can be identified are the Timetable Agent and the Student Agents: Problem Agent and Stakeholder Agents respectively. The Timetable Agent creates the initial timetable; in that it assigns each Student agent a set of slots which make up their timetable. The Student agents represent the preferences of each student in the system, and thus act on their behalf.

One way in which the system could be designed is to allow a student to advertise their time slot to all other agents in a system and accept the first acceptable swap. This would require the receiving student to determine whether the received timeslots are suitable according to their utility, and for the proposing student to determine whether an accepted proposal would maximise its utility. However, as discussed, this is a very inefficient system and is not suited for protecting student preferences. Therefore, this is not an acceptable way to design the system's communication protocol.

Another method would be to create an Advertiser Agent which acts as a second Problem Agent in the system and is responsible for handling all swaps in the system. This would hold an advertiser board, where slots are 'posted' by students and can then be accessed by other students to propose a swap. The Advertiser would act as a go-between in all student communication, as this protects student preferences and ensures validity within swaps. This would be an effective solution as it splits up 'Problem logic' and provides an effective way to accommodate swaps within the system.

However, in this system it was decided to have the Timetable Agent both create and ensure the integrity of the timetable – and so it is responsible for swaps within the system. This is in accordance with having a singular Problem agent which represents the problem scope.

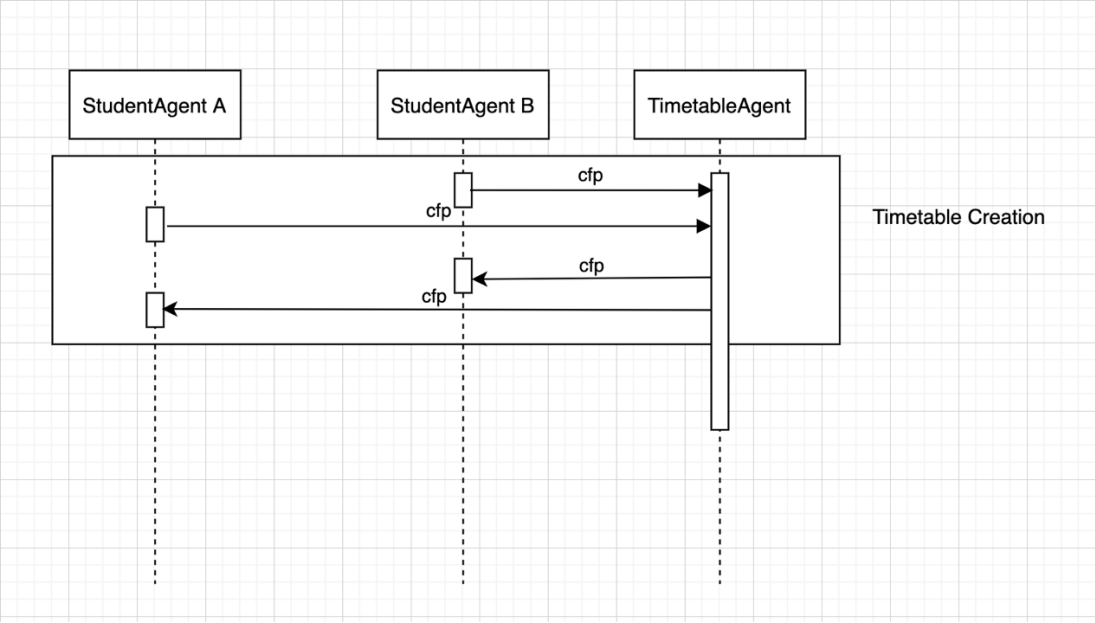

Figure 1. A sequence diagram that represents the Receive Timetable conversation.

The 'Receive Timetable' conversation involves two cfp messages from both Student Agents to the Timetable Agent, and one cfp message back from the Timetable Agent to each Student. This is

because the Student must make a request to be added to the list of Students, and the Timetable must then send the Student's timetable back to them.

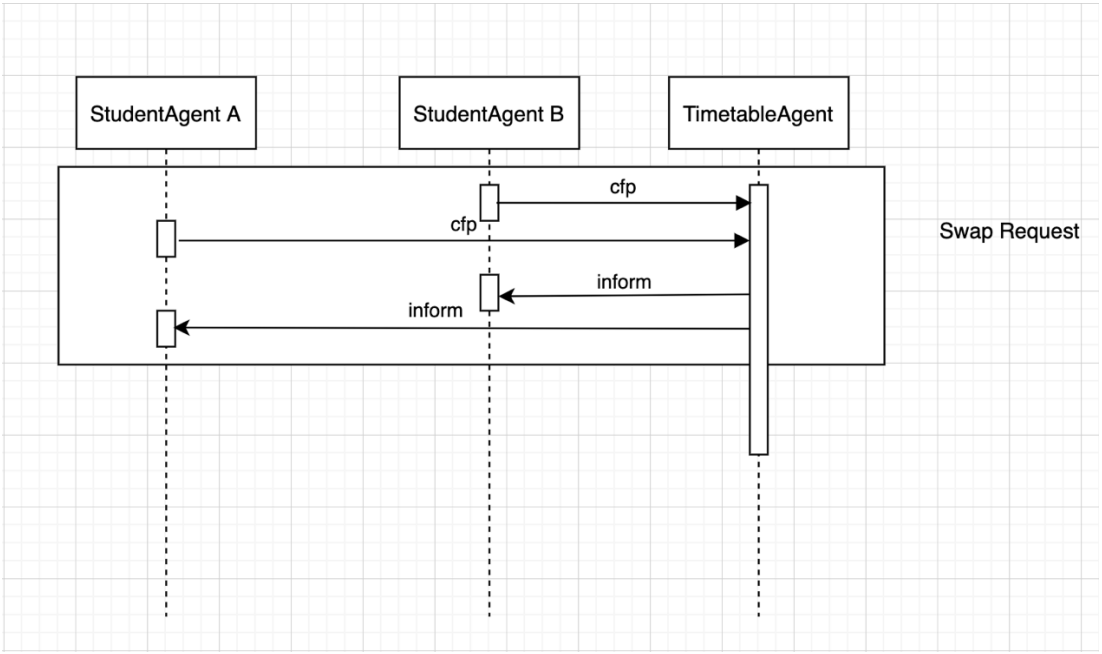

Figure 2. A sequence diagram that represents the Swap Request conversation.

The 'Swap Request' conversation involves two cfp messages from the Students to the Timetable Agent. This is the Student requesting the addition of their slot the 'Available Timeslots' list, and the Timetable Agent informing them this process has been completed.

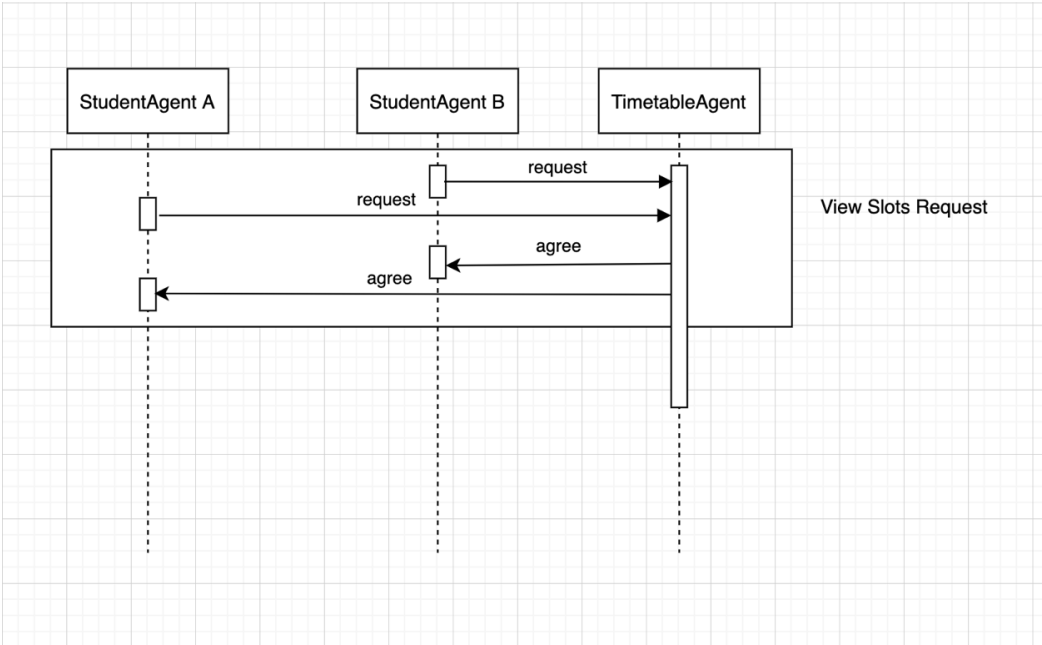

Figure 3. A sequence diagram representing the 'View Slots' conversation.

The 'View Slots' conversation involves the Students sending a request message to the Timetable Agent, and the Timetable Agent responding. This is the Student 'requesting' the list from the Timetable, and the Timetable Agent 'agreeing' in sending this to the Student.

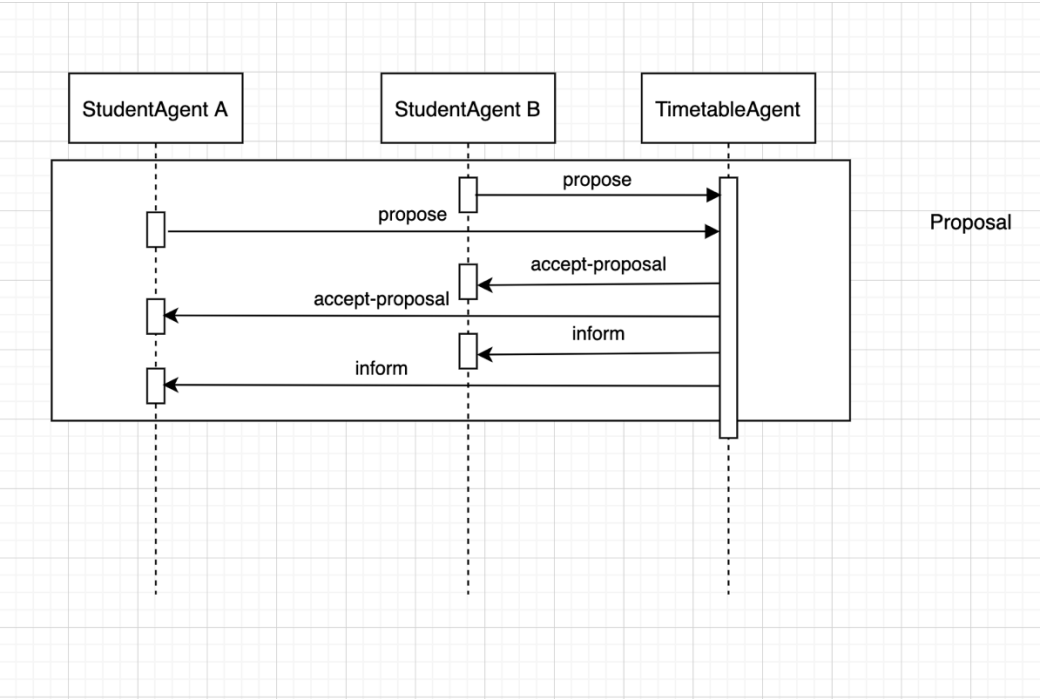

Figure 4. A sequence diagram that represents the 'Swap Proposal' conversation.

The 'Swap Proposal' conversation involves the Student sending a propose message to the Timetable Agent, and the Timetable Agent sending an accept-proposal message followed by an inform message. The Student sends the receiver they wish to propose a swap with, and the Timetable Agent sends back a message to confirm this swap.

1.3. Utility Function

The utility function for this system measures a student's 'level of satisfaction' for a given timeslot. This is broken down into levels according to the brief – cannot attend, would prefer not to attend, and would like to attend. These values are then given a 'score':

- Cannot attend has a score of -2
- Would prefer not to attend has a score of -1
- Would like to attend has a score of 1
- A score of 0 is given for neutral values, where the student has no particular preference.

This score is measured for each timeslot the student is given and is the basis for which a student will decide whether or not it wishes to make a swap. The utility score is given as a result of a utility function – the student's score can be totalled to determine how happy they are with their timetable overall.

Level	**Score**
Cannot attend	-2
Would prefer not to attend	-1
Would like to attend	1
Neutral	0

An alternative would be to give an integer value for if a slot aligns with a student's preferences. This would require an if statement which determines whether or not a given slot aligns with a student's

preferred day and start time/end time. However, this is a fairly unnuanced way to represent the problem – student's preferences should always be the primary goal of the utility function but an improvement in their situation is also an acceptable outcome. Therefore, this is not suitable for the problem scope.

Another alternative would be to have more factors which affect the utility score – in addition to the base score. For example, generally, the more days off in a week a student has, the happier they will be with their overall timetable and so this could give a 'boost' to a result. This was not implemented as it is outside of the given scope and would require careful consideration in terms of what effect each additional factor would have on the score (to ensure results weren't skewed) but would likely be a useful addition to the system if implemented further.

1.4. A strategy for the student agent that determines which requests to make, accept, reject

As described in 1.3, a student's evaluation of which requests it should make is largely dependent on the result of the utility function.

With the result of the utility function giving out a 'score' value, it is fairly simple to determine which exchange requests the Student Agent will either make or accept/reject from others. The student will evaluate their initial timetable in accordance with the utility function, and this will determine whether or not they will request a swap with the Timetable Agent. When receiving a timeslot back, its level of satisfaction will be compared against the current satisfaction level to determine whether or not a swap should be accepted.

On a lower level, a Student Agent will never request a swap for a timeslot that results in a positive number. The agent will only ever publish a swap in the pursuit of a timeslot that better matches their preference – and will therefore accept any change that will allow an increase in their satisfaction. For example, a swap from -2 to -1 or a swap from -1 to 0. This is handled cyclically, and a student will perform this process iteratively until the final tick of the system (which is capped to ensure the system does not run forever for the benefit of unsatisfied students who cannot find a swap). The Student, therefore, will never communicate to the Timetable Agent unless any timeslots in its timetable are unsatisfactory. Generally, a swap will not be made if the passed timeslot does not allow preferences – unless going from an 'unable to attend' to a 'would prefer not to attend'. This is the most extreme case of a swap being allowed – in theory this swap is not adhering to the student's preferences but is still increasing the overall utility of their timetable and resulting in a, while not optimal, better timetable for the student.

1.5. A metric to evaluate overall effectiveness

One way in which the overall effectiveness of the system can be tracked in terms of how well it satisfies student preference is to maintain a 'satisfaction' for each student. That is, a value that fluctuates depending on the level of satisfaction each student has with their modules. This is also suitable as a metric of how effective the utility function was in giving each student a suitable timetable. In practice, this is fairly straightforward to measure and calculate as it would simply require the creation and manipulation of a global 'happiness' value. This would be updated depending on the 'result' of each student's timetable. For example, if a student has three modules with timeslots they would prefer to attend, this can be given a numerical value i.e., +1 for every value and 3 overall. If a system is implemented this way, there would always be a set of expected numbers and so they could be given real-world associated 'scores' of satisfaction – for example 3 is a perfect score, but 2 would still likely fall in the bounds of an acceptable score. Each student's

happiness could then be interpreted individually. An example of a way these scores could be interpreted is as follows.

Score	Timetable Results	Level of Satisfaction
-3	Three modules the student is unhappy with.	Worst-case.
-2	Two modules the student is unhappy with and one they are neutral about.	Unhappy.
-1	One module the student is unhappy with and two they are neutral about.	Fairly Unhappy.
0	Three modules the student is neutral about.	Neutral.
1	One module the student is happy with and two they are neutral about.	Fairly Happy.
2	Two modules the student is happy with and one they are neutral about.	Happy.
3	Three modules the student is happy with.	Best-case.

If the system wasn't implemented in a ticker fashion, the time taken to have each student have their preferences matched could be measured as a metric of how 'effectively' the system matches student requirements. A student would send a message to the Timetable Agent when their preferences are matched. A potential issue could occur where a student continually requests a swap for a timeslot that no other student wants. However, this could be managed by having a micro ticker which closes the system after a certain number of repetitions of the student making a swap request. This would allow for another metric in seeing how many students didn't have their preferences matched at the end of the iteration. This would be a fairly effective way to evaluate the system, as it offers both an evaluation of the system itself on a macro level (how efficient it is) and in how 'well' the system matches student requirements. However, it is fairly limited in that it offers no nuance in its evaluation of matching student requirements – as described earlier, it would be more effective to use a scale to measure to which extent a student has had their preferences met than to measure it on a 'black and white' scale.

Finally, the metric implemented in this system involves tracking a 'global' happiness metric. This would be implemented in a similar manner to the local satisfaction level – however, this metric would give a better idea of the overall satisfaction across all students in the system. This is also a very simple metric to implement as it would simply require extending upon the student's satisfaction level (which is already calculated for use in interpreting timeslots) and adding all student satisfaction levels together. While an overall satisfaction level can be inferred from just using all local student satisfaction levels as the metric, this works much better as the system scales up. For example, it is not unrealistic that 100 students may have to be timetabled in a real-world scenario. Individually assessing each student's happiness level and assessing the overall level of satisfaction is fairly impractical and so it would be more effective to calculate this at a system level. As above, there is always a set number of expected results for this variable and so a 'category' of overall satisfaction can be inferred from the result.

Chapter 2 – Implementation

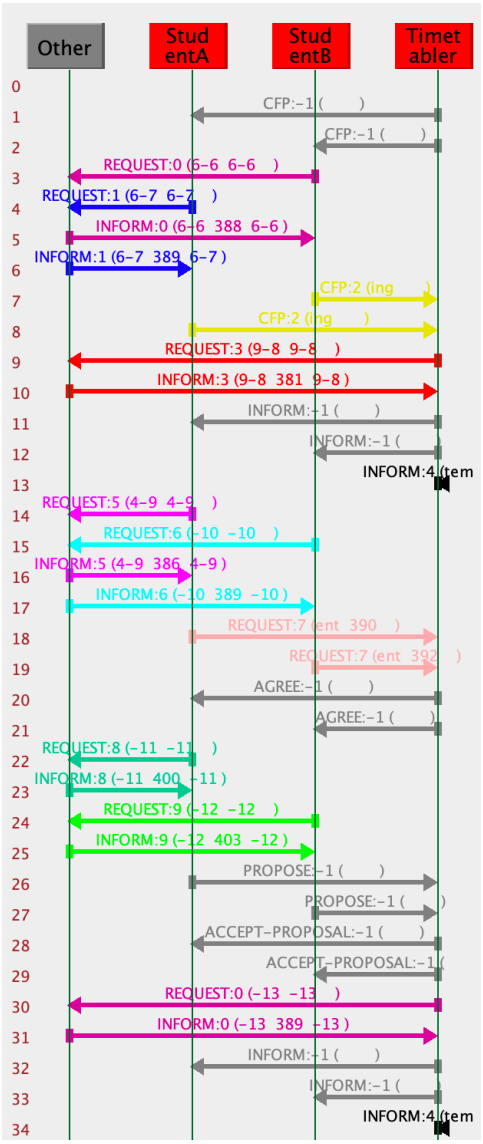

Figure 5. A screenshot showing the JADE sniffer and the relevant conversations in the system.

The above screenshot shows the JADE sniffer and the relevant conversations in which students can optimise their timetable in communication with the Timetable Agent. The conversations can be described as:

- A student requesting their addition to the timetable. This can be seen in the Timetable Agent's cfp message to both StudentA and StudentB. Note that this screenshot is missing the Students' initial cfp message to the Timetable Agent – this is because this is conducted on start-up and occurs before the sniffer has time to capture it.
- A student asking the Timetable Agent if they can send them their slot for a swap. When a student is unhappy with a timeslot they send a cfp message to the Timetable Agent with their Timeslot. The Timetable agent then sends an inform message back to the Student agent (which is inside its tick method) when the student's timeslot has been added to the list of open timeslots.

- A student asking if they can view the list of open timeslots in the system. A student sends a request method to the Timetable Agent. The Timetable Agent then sends an agree message back to the Student Agent with a list of all available timeslots.
- The student proposes a swap. The student initially sends a propose message to the Timetable Agent, which contains a SwapProposal instance (which implements AgentAction). The Timetable Agent then sends an accept-proposal message. An inform message is then visible, which is the standard tick message which represents a 'round'. The above process would then be repeated until the tick count reaches 10.

```
private int utilityFunction(Tutorial tutorial) {
    int level = 0;
    for (int i = 0; i < preferencesList.size(); i++) {
        if ((preferencesList.get(i).getStartTime() <= tutorial.getStartTime()) && (preferencesList.get(i).getEndTime()
                >= tutorial.getEndTime()) && preferencesList.get(i).getDay().equals(tutorial.getDay())) {
            switch (preferencesList.get(i).getLevel()) {
                case "Unable":
                    level = -2;
                    break;
                case "Prefer Not":
                    level = -1;
                    break;
                case "Would Like":
                    level = 1;
                    break;
            }
        }
    }
    return level;
}
```

Figure 6. A screenshot showing the utility function inside the Student Agent class.

The utilityFunction is where the Student Agent calculates its utility. A variable is set to hold the category of preference each timeslot the student holds has. An if statement determines if the student's timeslot falls under any of their categories of preference, and the timeslot's level is then run through a switch statement to get its score according to which preference it falls under.

```
//If result of utility is below 0, student needs to make their swap available
for(int i = 0; i < timetableList.size(); i++) {
    int util = utilityFunction(timetableList.get(i));
    satisfactionLevel = 0;
    satisfactionLevel = satisfactionLevel + util;
    if(util < 0) {
        swapWanted = true;
    }
}
if(swapWanted == true) {
    DFAgentDescription template = new DFAgentDescription();
    ServiceDescription desc = new ServiceDescription();
    ACLMessage cfp = new ACLMessage(ACLMessage.REQUEST);
    desc.setType("TimetablingAgent");
    template.addServices(desc);
    try {
        DFAgentDescription[] result = DFService.search(myAgent, template);
        if (result.length > 0) {
            cfp.addReceiver(result[0].getName());
        }
    } catch (FIPAException fe) {
        fe.printStackTrace();
    }
    cfp.setContent("requestSlots");
    cfp.setConversationId("TimetablingAgent");
    cfp.setReplyWith("cfp" + System.currentTimeMillis()); // Unique value
    myAgent.send(cfp);
```

Figure 7. Determines whether a student should request a swap, inside handleTimetablignAgentMessage in Student Agent class.

The student's list of timeslots is run through the utility function. If the result of the utility function is less than 0, then a Boolean called swapWanted is set to true. In this case, the Student sends the requestSlots cfp message detailed in 1.2.

```
if (msg != null) {
    ContentElement ce = null;
    try {
        ce = getContentManager().extractContent(msg);
        if (ce instanceof AvailableTimeslots) {
            AvailableTimeslots availableTimeslots = (AvailableTimeslots) ce;

            int slotWanted = -1;
            for (int i = 0; i < availableTimeslots.getTimeslots().size(); i++) {
                for (int j = 0; j < timetableList.size(); j++) {
                    //If slots are compatible i.e. of the same module
                    if (availableTimeslots.getTimeslots().get(i).getModuleName().equals(timetableList.get(j).getModuleName())) {

                        int swappedSlotSatisfactionLevel = utilityFunction(availableTimeslots.getTimeslots().get(i));
                        int currentSatisfactionLevel = utilityFunction(timetableList.get(j));

                        if (swappedSlotSatisfactionLevel > currentSatisfactionLevel) {
                            slotWanted = i;
                        }
                    }
                }
            }

            System.out.println("The slot wanted by the student (i.e. index of for loop) is slot " + slotWanted);
```

Figure 8. A screenshot which shows the potential swap being compared to the current swap inside the handleSwapWanted method in the Student Agent class.

The above code is inside the handleSwapWanted method and determines if a swap should be made. A loop is performed over all timeslots, and if an available timeslot's name is the same as the current timeslot's name the result of its utility function is compared against the current result. If the potential timeslot is better for the student than the current, a value called slotWanted is set to the index of that timeslot.

```
if (slotWanted != -1) {
    SwapProposal proposal = new SwapProposal();
    proposal.setTimeslotOwner(availableTimeslots.getTimeslots().get(slotWanted).getAttendee());
    proposal.setTimeslot(availableTimeslots.getTimeslots().get(slotWanted));
    proposal.setTimeslotReceiver(myAgent.getAID());

    ACLMessage requestSwapWithTimetablerMsg = new ACLMessage(ACLMessage.PROPOSE);
    DFAgentDescription template = new DFAgentDescription();
    ServiceDescription desc = new ServiceDescription();

    desc.setType("TimetablingAgent");
    template.addServices(desc);
    try {
        DFAgentDescription[] result = DFService.search(myAgent, template);
        if (result.length > 0) {
            requestSwapWithTimetablerMsg.addReceiver(result[0].getName());
        }
    } catch (FIPAException fe) {
        fe.printStackTrace();
    }
    requestSwapWithTimetablerMsg.setLanguage(codec.getName());
    requestSwapWithTimetablerMsg.setOntology(timetableOntology.getName());

    Action request = new Action();
    request.setAction(proposal);
    request.setActor(proposal.getTimeslotReceiver());
    try {
        System.out.println("\nStudent is requesting a swap with the Timetabling Agent...");
        // Let JADE convert from Java objects to string
        getContentManager().fillContent(requestSwapWithTimetablerMsg, request); // send the wrapper object
        send(requestSwapWithTimetablerMsg);
```

Figure 9. A screenshot showing a swap proposal inside handleSwapWanted inside the Student Agent class.

The above code shows the code which handles swap proposals in the system. An instance of the SwapProposal class is created, and a Propose message is created. The message's receiver is set to the propose. As a proposal is an Action, a request is set to get the Agent ID of the 'receiver' student and the request message is sent.

```
public class StudentPreferences {
    String level;
    String day;
    int startTime;
    int endTime;

    public String getLevel() { return level; }

    public void setLevel(String level) { this.level = level; }

    public String getDay() { return day; }

    public void setDay(String day) { this.day = day; }

    public int getStartTime() { return startTime; }

    public void setStartTime(int startTime) { this.startTime = startTime; }

    public int getEndTime() { return endTime; }

    public void setEndTime(int endTime) { this.endTime = endTime; }

    public StudentPreferences(String level, String day, int sTime, int eTime) {
        this.level = level; //Unable, prefer, would like
        this.day = day;
        startTime = sTime;
        endTime = eTime;
    }
    public StudentPreferences() {

    }
```

Figure 10. A screenshot showing the Student Preferences class.

The StudentPreferences class is where student preferences are held – this is intentionally abstracted to hide preferences from the rest of the system.

```
private class manageStudents extends CyclicBehaviour {
    public void action() {
        MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
        ACLMessage msg = myAgent.receive(mt);
        if (msg != null) {
            if (msg.getContent().equals("add me to list")) {
                studentList.add(msg.getSender());
                System.out.println("Student added to timetable.");
            }
            else {
                try {
                    ContentElement ce = null;
                    ce = getContentManager().extractContent(msg);
                    if (ce instanceof Timeslot) {
                        Timeslot owns = (Timeslot) ce;
                        availableTimeslots.getTimeslots().add(owns.getTutorial());
                        System.out.print("\nTimetabler Agent added slot: " + owns.getTutorial().getModuleName() + " at "
                                + owns.getTutorial().getStartTime());
                    }
                }
                catch (CodecException ce) {
                    ce.printStackTrace();
                } catch (OntologyException oe) {
                    oe.printStackTrace();
                }
            }
        }
    }
}
```

Figure 11. A screenshot showing the manageStudents method inside the Timetable Agent class.

```
private class updateSlot extends CyclicBehaviour {
    public void action() {
        MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROPOSAL);
        ACLMessage msg = myAgent.receive(mt);

        if (msg != null) {
            ContentElement ce = null;
            try {
                ce = getContentManager().extractContent(msg);
                if (ce instanceof Timeslot) {
                    Timeslot newSlot = (Timeslot) ce;
                    for (int i = 0; i < timetableList.size(); i++) {
                        if (timetableList.get(i).getModuleName().equals((newSlot.getTutorial().getModuleName()))) {
                            timetableList.remove(i);
                            timetableList.add(newSlot.getTutorial());
                            System.out.println("Student " + myAgent.getName() + " has swapped for the " +
                                    newSlot.getTutorial().getDay() + " " + newSlot.getTutorial().getModuleName() + " slot." );
                        }
                    }
                }
            } catch (CodecException | OntologyException e) {
                e.printStackTrace();
            }
        }
    }
```

Figure 12. A screenshot showing the updateSlot method inside the Student Agent class.

This is where it is ensured that students attend only one tutorial for each module. A ContentManager creates an instance of a Timeslot, and updateSlot ensures slots are ‘properly’ swapped i.e. ownership is exchanged.

Chapter 3 – Testing

Test Name	Problem	Input	Original Happiness	Output Happiness
TestOne()	One module, two timeslots and two students.	Two students with timetables that do not match their preferences.	-2	0
TestTwo()	Two modules with two timeslots each and two students.	Two students who are unhappy with both of their slots.	-2 for both timeslots	0

Chapter 4 – Evaluation

1.1 How will the effectiveness of your system change as the problem becomes more difficult?

As the problem grows, there are more students which in turn means it is harder to please everyone. The solution to this is that the system would require more rounds to ensure the most acceptable situation for every student, which could lead to endless iterations over the timetable. An issue also arises as utility becomes more abstract as the number of students grow - naturally, one student's happiness is much less reflective of the general happiness of the student population when there are 100 students compared to 3 students.

A system where the student has to compare against every other student becomes less optimal over time - naturally, it will take longer to communicate with 99 students than 2 other students and determine which swaps are the best for them. This process could potentially be carried out by all other students in the system, with no regards for each other's utility.

In effect, this is the major flaw of the system - the utility of other students has no bearing on the choices of students in the system. This can, in theory, lead to a perpetual scenario where students continually make choices that negatively impact other student's happiness. While each individual student has their happiness reflected, there is nothing to account for the overall happiness of the students. In a system where only, for example, four students have to communicate, this is less of an issue. However, it is easy to see how this system can result in increasing complexity and require a far greater number of 'rounds' as the student population increases. This does not make the system less effective per se but can lead to a less effective result for the student population at large if something like a cut-off for the number of rounds is implemented to ensure the system does not perpetually iterate over itself. For example, consider the following scenario:

- Student A would prefer not to attend their slot.
- Student B would also prefer not to attend their slot, and so they request a swap.
- Student B swaps their slot for Student A's slot and both students have their preferred slots, leading to a 'perfect' scenario in their cases.
- Student C is neutral about their slot. However, this student is unable to swap their slot.

This would result in a situation where there is always one neutral student who continually posts their slot to the advertisement board, with no options left to increase their happiness. If this scenario is not dealt with, the system would run in perpetuity with no possible solution to the problem. Therefore, a system must be put in place to ensure the program has 'stop-gaps' so that it achieves a solution to the best of its ability. This problem can be magnified on a much higher scale as the student base increases.

1.2 What are the advantages and disadvantages of taking a multi-agent systems approach to this problem?

The primary advantage of a multi-agent system is that it allows for each stakeholder in an environment to be truly represented. That is to say that the inherent way in which multi-agent systems are developed allows for a 'real' representation of every stakeholder in a given problem to have their needs considered. Agents act autonomously to achieve their goals and according to their behaviours, and so their interests are inherently accounted for when interacting with the system. Having a Problem Agent allows for the 'management' of the problem, and the assurance that stakeholders are treated fairly, and the solution maintains valid.

However, while it is true that everyone has their interests represented, agents do not take other agents' happiness into account. As explained earlier, this can lead to a 'free for all' situation where students blindly make decisions solely based on their own utility. While this satisfies that individual student's utility, this can negatively impact other student's happiness and in turn decrease the overall happiness of the system.

1.3 In light of 1 and 2, suggest and justify an improvement to your system.

Based on the issues identified with the current system, a major development would be to have overall student happiness play a larger role in the timetabling decisions made in the system. This could take many forms, but the basis is to have a student swap be dependent on its affect to the global happiness of students rather than the local happiness of the students involved in the swap. This represents a positive change in the 'nature' of students, as while they have their own interests, the system should ultimately have the goal of providing a timetable which satisfies the maximum number of students – rather than having the goal of making the current two students (the owner and receiver) happy.

This idea can be extended further – Student Agents can remain independent but can be grouped into a global 'student body'. This could allow timetables to contrast their own timetables in a collaborative manner with the Timetable Agent, or at the very least allow them to request a move of a particular timeslot (Urquhart & Powers, 2019). The introduction of a Lecturer Agent could allow for the construction of a timetable from the ground up, with this agent being another stakeholder in the system. The Timetable Agent would continue to act as the Problem Agent in this scenario, and Lecturer preferences would hold more 'weight' than Student Agents. This could allow for a truly collaborative environment without any real constraints beyond the hard preferences set by the Lecturer for the individual tutorials – dependent on real-world variables such as room availability etc. – and would ensure a maximum level of student and lecturer satisfaction.

At the very least, the introduction of a global happiness variable which is either used as the basis for swaps or holds some level of influence on whether or not a swap is 'allowed' under the system would likely encourage a higher level of happiness for all students in the system.

References

Urquhart, N., & Powers, S. T. (2019). An Agent Based Technique for Improving Multi-stakeholder Optimisation Problems. *International Conference on Practical Applications of Agents and Multi-Agent Systems*, 285–289.

Appendices

```
 1 import java.util.ArrayList;
 2 import java.util.List;
 3 import jade.content.Concept;
 4 import jade.content.ContentElement;
 5 import jade.content.lang.Codec;
 6 import jade.content.lang.Codec.CodecException;
 7 import jade.content.lang.sl.SLCodec;
 8 import jade.content.onto.Ontology;
 9 import jade.content.onto.OntologyException;
10 import jade.content.onto.basic.Action;
11 import jade.core.AID;
12 import jade.core.Agent;
13 import jade.core.behaviours.CyclicBehaviour;
14 import jade.core.behaviours.TickerBehaviour;
15 import jade.core.behaviours.WakerBehaviour;
16 import jade.domain.DFService;
17 import jade.domain.FIPAException;
18 import jade.domain.FIPAAgentManagement.
   DFAgentDescription;
19 import jade.domain.FIPAAgentManagement.
   ServiceDescription;
20 import jade.lang.acl.ACLMessage;
21 import jade.lang.acl.MessageTemplate;
22 import timetable_ontology.*;
23
24 public class TimetableAgent extends Agent{
25     private Codec codec = new SLCodec();
26     private Ontology ontology = TimetableOntology.
   getInstance();
27
28     List<AID> studentList = new ArrayList<AID>();
29     AvailableTimeslots availableTimeslots = new
   AvailableTimeslots();
30     ArrayList<SwapProposal> proposals = new
   ArrayList<SwapProposal>();
31
32     int tickCount = 0;
33
34     int satisfactionLevel = 0;
35     int globalSatisfaction = 0;
36
37     protected void setup() {
38         getContentManager().registerLanguage(codec
   );
```

```
39          getContentManager().registerOntology(
    ontology);
40
41          DFAgentDescription dfd = new
    DFAgentDescription();
42          dfd.setName(getAID());
43          ServiceDescription sd = new
    ServiceDescription();
44          sd.setType("TimetablingAgent");
45          sd.setName("TimetablingAgent");
46          dfd.addServices(sd);
47          try {
48              DFService.register(this, dfd);
49          } catch (FIPAException fe) {
50              fe.printStackTrace();
51          }
52          addBehaviour(new WakerBehaviour(this, 10000
    ) {
53              protected void onWake() {
54                  System.out.println("There are " +
    studentList.size() + " student agents in the system
    ");
55
56                  //testOne();
57                  testTwo();
58                  //testThree();
59              }
60          });
61          addBehaviour(new TickerBehaviour(this, 6000
    ) {
62              protected void onTick() {
63                  tick();
64              }
65          });
66          this.addBehaviour(new manageStudents());
67          addBehaviour(new handleSwapRequest());
68          addBehaviour(new handleSwapProposal());
69          addBehaviour(new calcGlobalSatisfaction());
70      }
71
72
73
74      private void testTwo() {
75          for (int i = 0; i < studentList.size(); i
```

```
 75 ++) {
 76
 77             ACLMessage multimsg = new ACLMessage(
    ACLMessage.CFP);
 78             multimsg.addReceiver(studentList.get(i
    ));
 79             multimsg.setLanguage(codec.getName());
 80             multimsg.setOntology(ontology.getName
    ());
 81
 82             ACLMessage softwaremsg = new
    ACLMessage(ACLMessage.CFP);
 83             softwaremsg.addReceiver(studentList.
    get(i));
 84             softwaremsg.setLanguage(codec.getName
    ());
 85             softwaremsg.setOntology(ontology.
    getName());
 86
 87             Tutorial multiAgents = new Tutorial();
 88             Tutorial software = new Tutorial();
 89
 90             switch (i) {
 91                 case 0 → {
 92                     multiAgents.setAttendee(
    studentList.get(i));
 93                     multiAgents.setDay("Tuesday");
 94                     multiAgents.setModuleName("
    MultiAgents");
 95                     multiAgents.setModuleNo("
    SET1011");
 96                     multiAgents.setCampus("
    Merchiston");
 97                     multiAgents.setLecturer("Aaron
    ");
 98                     multiAgents.setStartTime(1200
    );
 99                     multiAgents.setEndTime(1300);
100
101                     software.setAttendee(
    studentList.get(i));
102                     software.setDay("Tuesday");
103                     software.setModuleName("
    Software");
```

```
104                         software.setModuleNo("SET10101
    ");
105                         software.setCampus("Merchiston
    ");
106                         software.setLecturer("Aaron");
107                         software.setStartTime(1400);
108                         software.setEndTime(1500);
109                     }
110                     case 1 → {
111                         multiAgents.setAttendee(
    studentList.get(i));
112                         multiAgents.setDay("Friday");
113                         multiAgents.setModuleName("
    MultiAgents");
114                         multiAgents.setModuleNo("
    SET1011");
115                         multiAgents.setCampus("
    Merchiston");
116                         multiAgents.setLecturer("Aaron
    ");
117                         multiAgents.setStartTime(1500
    );
118                         multiAgents.setEndTime(1600);
119
120                         software.setAttendee(
    studentList.get(i));
121                         software.setDay("Friday");
122                         software.setModuleName("
    Software");
123                         software.setModuleNo("SET10101
    ");
124                         software.setCampus("Merchiston
    ");
125                         software.setLecturer("Aaron");
126                         software.setStartTime(1400);
127                         software.setEndTime(1500);
128
129                     }
130                 }
131                 Timeslot multi = new Timeslot();
132                 multi.setTimeslotOwner(studentList.get
    (i));
133                 multi.setTutorial(multiAgents);
134
```

```
135             Timeslot soft = new Timeslot();
136             soft.setTimeslotOwner(studentList.get(
    i));
137             soft.setTutorial(software);
138 
139             try {
140                 getContentManager().fillContent(
    multimsg, multi);
141                 send(multimsg);
142 
143                 getContentManager().fillContent(
    softwaremsg, soft);
144                 send(softwaremsg);
145             } catch (CodecException ce) {
146                 ce.printStackTrace();
147             } catch (OntologyException oe) {
148                 oe.printStackTrace();
149             }
150         }
151 
152     }
153 
154 
155 
156     private void tick() {
157         if (tickCount < 5) {
158             System.out.println("\nTick.");
159             for(int i = 0; i < proposals.size(); i
    ++) {
160                 for(int j = 0; j < proposals.size
    (); j++) {
161                     if(proposals.get(i).
    getTimeslotOwner().equals(proposals.get(j).
    getTimeslotReceiver())) {
162                         ACLMessage msg = new
    ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
163                         msg.addReceiver(proposals.
    get(i).getTimeslotReceiver());
164                         msg.setLanguage(codec.
    getName());
165                         msg.setOntology(ontology.
    getName());
166                         // Prepare the content.
167                         Timeslot owns = new
```

```
167 Timeslot();
168                         owns.setTimeslotOwner(
    proposals.get(i).getTimeslotOwner());
169                         owns.setTutorial(proposals
    .get(i).getTimeslot());
170                         try {
171                             // Let JADE convert
    from Java objects to string
172                             getContentManager().
    fillContent(msg, owns);
173                             System.out.println("
    Slot sent to Timetable agent.");
174                             send(msg);
175                         } catch (CodecException ce
    ) {
176                             ce.printStackTrace();
177                         } catch (OntologyException
     oe) {
178                             oe.printStackTrace();
179                         }
180                     }
181                 }
182             }
183             proposals.removeAll(proposals);
184 
185             DFAgentDescription template = new
    DFAgentDescription();
186             ServiceDescription desc = new
    ServiceDescription();
187             ACLMessage cfp = new ACLMessage(
    ACLMessage.INFORM);
188             desc.setType("TimetablingAgent");
189             template.addServices(desc);
190             try {
191                 DFAgentDescription[] result =
    DFService.search(this, template);
192                 if (result.length > 0) {
193                     cfp.addReceiver(result[0].
    getName());
194                 }
195             } catch (FIPAException fe) {
196                 fe.printStackTrace();
197             }
198             cfp.setContent("tickInform");
```

```
199             for (int i = 0; i < studentList.size
    (); i++) {
200                 ACLMessage inform = new ACLMessage
    (ACLMessage.INFORM);
201                 inform.addReceiver(studentList.get
    (i));
202                 inform.setContent("tickInform");
203                 this.send(inform);
204             }
205             cfp.setConversationId("timetable-
    system");
206             this.send(cfp);
207 
208             tickCount++;
209 
210         } else {
211 
212             getSatisfaction();
213 
214 
215             System.out.println("Deconstructing..."
    );
216             DFAgentDescription template = new
    DFAgentDescription();
217             ServiceDescription desc = new
    ServiceDescription();
218             ACLMessage cfp = new ACLMessage(
    ACLMessage.INFORM);
219 
220             desc.setType("TimetablingAgent");
221             template.addServices(desc);
222             try {
223                 DFAgentDescription[] result =
    DFService.search(this, template);
224                 if (result.length > 0) {
225                     cfp.addReceiver(result[0].
    getName());
226                 }
227             } catch (FIPAException fe) {
228                 fe.printStackTrace();
229             }
230 
231             cfp.setContent("takedownRequest");
232             cfp.setConversationId("timetable-
```

```
232 system");
233             this.send(cfp);
234 
235             this.doDelete();
236         }
237     }
238 
239     private void getSatisfaction()  {
240         for (int i = 0; i < studentList.size(); i
    ++) {
241 
242             DFAgentDescription template = new
    DFAgentDescription();
243             ServiceDescription desc = new
    ServiceDescription();
244             ACLMessage cfp = new ACLMessage(
    ACLMessage.REQUEST);
245 
246             template.addServices(desc);
247             cfp.addReceiver(studentList.get(i));
248             cfp.setContent("satisfactionRequest");
249             this.send(cfp);
250         }
251     }
252 
253     private class calcGlobalSatisfaction extends
    CyclicBehaviour {
254         public void action() {
255             MessageTemplate mt = MessageTemplate.
    MatchPerformative(ACLMessage.PROPAGATE);
256             ACLMessage msg = myAgent.receive(mt);
257             if (msg ≠ null) {
258                 int studentSatisfaction = Integer.
    parseInt(msg.getContent());
259                 globalSatisfaction =
    globalSatisfaction + studentSatisfaction;
260             }
261         }
262     }
263 
264     // Put agent clean-up operations here
265     protected void takeDown() {
266         // Deregister from the yellow pages
267 
```

```
268         System.out.print("The global happiness of
    the system is: " + globalSatisfaction+"\n");
269 
270         try {
271             for (int i = 0; i < studentList.size
    (); i++) {
272 
273                 DFAgentDescription template = new
    DFAgentDescription();
274                 ServiceDescription desc = new
    ServiceDescription();
275                 ACLMessage inform = new ACLMessage
    (ACLMessage.INFORM);
276                 template.addServices(desc);
277                 inform.addReceiver(studentList.get
    (i));
278                 inform.setContent("takedownRequest
    ");
279                 this.send(inform);
280             }
281             DFService.deregister(this);
282             } catch (FIPAException fe) {
283                 fe.printStackTrace();
284             }
285             System.out.println("Timetabler agent
    terminating.");
286         }
287 
288     private class manageStudents extends
    CyclicBehaviour {
289         public void action() {
290             MessageTemplate mt = MessageTemplate.
    MatchPerformative(ACLMessage.CFP);
291             ACLMessage msg = myAgent.receive(mt);
292             if (msg ≠ null) {
293                 if (msg.getContent().equals("add
    me to list")) {
294                     studentList.add(msg.getSender
    ());
295                     System.out.println("Student
    added to timetable.");
296                 }
297                 else {
298                     try {
```

```
299                             ContentElement ce = null;
300                             ce = getContentManager().
    extractContent(msg);
301                             if (ce instanceof Timeslot
    ) {
302                                 Timeslot owns = (
    Timeslot) ce;
303                                 availableTimeslots.
    getTimeslots().add(owns.getTutorial());
304                                 System.out.print("\n
    Timetabler Agent added slot: " + owns.getTutorial
    ().getModuleName() + " at "
305                                         + owns.
    getTutorial().getStartTime());
306                             }
307                         }
308                         catch (CodecException ce) {
309                             ce.printStackTrace();
310                         } catch (OntologyException oe
    ) {
311                             oe.printStackTrace();
312                         }
313                     }
314                 }
315             }
316         }
317 
318         private class handleSwapRequest extends
    CyclicBehaviour{
319             public void action() {
320                 MessageTemplate mt = MessageTemplate.
    MatchPerformative(ACLMessage.REQUEST);
321                 ACLMessage msg = myAgent.receive(mt);
322 
323                 if(msg =/= null) {
324                     if(msg.getContent().equals("
    requestSlots")) {
325                         ACLMessage reply = new
    ACLMessage(ACLMessage.AGREE);
326 
327                         reply.setLanguage(codec.
    getName());
328                         reply.setOntology(ontology.
    getName());
```

```
329                     reply.addReceiver(msg.
    getSender());
330                     try {
331                         AvailableTimeslots
    slotPredicate = new AvailableTimeslots();
332                         slotPredicate.setTimeslots
    (availableTimeslots.getTimeslots());
333                         getContentManager().
    fillContent(reply, slotPredicate);
334                         send(reply);
335                         System.out.println("Sent
    reply.");
336                     } catch (CodecException e) {
337                         e.printStackTrace();
338                     } catch (OntologyException e
    ) {
339                         e.printStackTrace();
340                     }
341                 }
342             }
343         }
344     }
345 
346     private class handleSwapProposal extends
    CyclicBehaviour{
347         public void action() {
348 
349             MessageTemplate mt = MessageTemplate.
    MatchPerformative(ACLMessage.PROPOSE);
350             ACLMessage msg = myAgent.receive(mt);
351 
352             if(msg ≠ null) {
353                 try {
354                     ContentElement ce = null;
355                     ce = getContentManager().
    extractContent(msg);
356                     if(ce instanceof Action) {
357 
358                         Concept action = ((Action
    ) ce).getAction();
359                         if (action instanceof
    SwapProposal) {
360                             SwapProposal
    timeslotOwner = (SwapProposal) action;
```

```
361                                 proposals.add(
    timeslotOwner);
362                                 System.out.println("
    There has been a swap request for " +
    timeslotOwner.getTimeslot().getModuleName() + "\n
    from student " + timeslotOwner.getTimeslotOwner().
    getName());
363                             }
364                         }
365                     }
366                     catch (CodecException ce) {
367                         ce.printStackTrace();
368                     } catch (OntologyException oe) {
369                         oe.printStackTrace();
370                     }
371                 }
372             }
373         }
374 }
375
```

```java
package timetable_ontology;

import jade.content.Predicate;
import jade.core.AID;

public class Timeslot implements Predicate {

    private static final long serialVersionUID = 1L;
    private AID timeslotOwner;
    private Tutorial tutorial;

    public AID getTimeslotOwner() {
        return timeslotOwner;
    }

    public void setTimeslotOwner(AID timeslotOwner) {
        this.timeslotOwner = timeslotOwner;
    }

    public Tutorial getTutorial() {
        return tutorial;
    }

    public void setTutorial(Tutorial tutorial) {
        this.tutorial = tutorial;
    }
}

```

```java
package timetable_ontology;

import jade.content.Concept;
import jade.core.AID;

public class Tutorial implements Concept {

    AID attendee;
    String moduleName;
    String moduleNo;
    String campus;
    String lecturer;
    String day;
    int startTime;
    int endTime;

    public Tutorial() {

    }

    public AID getAttendee() {
        return attendee;
    }

    public void setAttendee(AID attendee) {
        this.attendee = attendee;
    }

    public String getModuleName() {
        return moduleName;
    }

    public void setModuleName(String moduleName) {
        this.moduleName = moduleName;
    }

    public String getModuleNo() {
        return moduleNo;
    }

    public void setModuleNo(String moduleNo) {
        this.moduleNo = moduleNo;
    }

```

```
45      public String getCampus() {
46          return campus;
47      }
48
49      public void setCampus(String campus) {
50          this.campus = campus;
51      }
52
53      public String getLecturer() {
54          return lecturer;
55      }
56
57      public void setLecturer(String lecturer) {
58          this.lecturer = lecturer;
59      }
60
61      public String getDay() {
62          return day;
63      }
64
65      public void setDay(String day) {
66          this.day = day;
67      }
68
69      public int getStartTime() {
70          return startTime;
71      }
72
73      public void setStartTime(int startTime) {
74          this.startTime = startTime;
75      }
76
77      public int getEndTime() {
78          return endTime;
79      }
80
81      public void setEndTime(int endTime) {
82          this.endTime = endTime;
83      }
84  }
85
```

```
 1 import jade.content.onto.basic.Action;
 2 import jade.core.Agent;
 3 import jade.core.behaviours.Behaviour;
 4 import jade.core.behaviours.CyclicBehaviour;
 5 import jade.domain.DFService;
 6 import jade.domain.FIPAException;
 7 import jade.domain.FIPAAgentManagement.
   DFAgentDescription;
 8 import jade.domain.FIPAAgentManagement.
   ServiceDescription;
 9 import jade.lang.acl.ACLMessage;
10 import jade.lang.acl.MessageTemplate;
11 import timetable_ontology.AvailableTimeslots;
12 import timetable_ontology.SwapProposal;
13 import timetable_ontology.Timeslot;
14 import timetable_ontology.TimetableOntology;
15 import timetable_ontology.Tutorial;
16
17 import java.util.ArrayList;
18 import java.util.List;
19
20 import jade.content.ContentElement;
21 import jade.content.lang.Codec;
22 import jade.content.lang.Codec.CodecException;
23 import jade.content.lang.sl.SLCodec;
24 import jade.content.onto.Ontology;
25 import jade.content.onto.OntologyException;
26 import jade.core.AID;
27
28 public class StudentAgent extends Agent {
29     List<Tutorial> timetableList = new ArrayList<
   Tutorial>();
30     List<StudentPreferences> preferencesList = new
   ArrayList<StudentPreferences>();
31     private final Codec codec = new SLCodec();
32     private final Ontology timetableOntology =
   TimetableOntology.getInstance();
33     private int satisfactionLevel = 0;
34
35     public int getSatisfactionLevel() {
36         return satisfactionLevel;
37     }
38
39     protected void setup() {
```

```
40          Object[] args = getArguments();
41          if (args ≠ null && args.length > 0) {
42              StudentPreferences preference = new
   StudentPreferences();
43              preference = (StudentPreferences) args[
   0];
44              preferencesList.add(preference);
45          }
46          getContentManager().registerLanguage(codec
   );
47          getContentManager().registerOntology(
   timetableOntology);
48
49          DFAgentDescription dfd = new
   DFAgentDescription();
50          dfd.setName(getAID());
51          ServiceDescription sd = new
   ServiceDescription();
52          sd.setType("timetabling");
53          sd.setName("student");
54          dfd.addServices(sd);
55          try {
56              DFService.register(this, dfd);
57          } catch (FIPAException fe) {
58              fe.printStackTrace();
59          }
60          addBehaviour(new handleTimetableReceival
   ());
61          addBehaviour(new timetableRegistration());
62          addBehaviour(new
   handleTimetablingAgentMessage());
63          addBehaviour(new handleSwapWanted());
64          addBehaviour(new updateSlot());
65          addBehaviour(new giveSatisfaction());
66      }
67
68      // Put agent clean-up operations here
69      protected void takeDown() {
70          // Deregister from the yellow pages
71          try {
72              DFService.deregister(this);
73          } catch (FIPAException fe) {
74              fe.printStackTrace();
75          }
```

```
 76     }
 77
 78     private int utilityFunction(Tutorial tutorial
    ) {
 79         int level = 0;
 80         for (int i = 0; i < preferencesList.size
    (); i++) {
 81             if ((preferencesList.get(i).
    getStartTime() ⩽ tutorial.getStartTime()) && (
    preferencesList.get(i).getEndTime()
 82                     ⩾ tutorial.getEndTime()) &&
    preferencesList.get(i).getDay().equals(tutorial.
    getDay())) {
 83                 switch (preferencesList.get(i).
    getLevel()) {
 84                     case "Unable":
 85                         level = -2;
 86                         break;
 87                     case "Prefer Not":
 88                         level = -1;
 89                         break;
 90                     case "Would Like":
 91                         level = 1;
 92                         break;
 93                 }
 94             }
 95         }
 96         return level;
 97     }
 98
 99     private class timetableRegistration extends
    Behaviour {
100         public void action() {
101             DFAgentDescription dfd = new
    DFAgentDescription();
102             ServiceDescription sd = new
    ServiceDescription();
103             ACLMessage cfp = new ACLMessage(
    ACLMessage.CFP);
104             sd.setType("TimetablingAgent");
105             dfd.addServices(sd);
106             try {
107                 DFAgentDescription[] result =
    DFService.search(myAgent, dfd);
```

```
108                 if (result.length > 0) {
109                     cfp.addReceiver(result[0].
    getName());
110                 }
111             } catch (FIPAException fe) {
112                 fe.printStackTrace();
113             }
114             cfp.setContent("add me to list");
115             cfp.setConversationId("timetabling");
116             cfp.setReplyWith("cfp" + System.
    currentTimeMillis()); // Unique value
117             myAgent.send(cfp);
118         }
119 
120         public boolean done() {
121             return true;
122         }
123     }
124 
125     private class handleTimetableReceival extends
    CyclicBehaviour {
126         public void action() {
127             MessageTemplate mt = MessageTemplate.
    MatchPerformative(ACLMessage.CFP);
128             ACLMessage msg = myAgent.receive(mt);
129             if (msg ≠ null) {
130                 try {
131                     ContentElement ce = null;
132                     ce = getContentManager().
    extractContent(msg);
133                     if (ce instanceof Timeslot) {
134                         Timeslot slot = (Timeslot
    ) ce;
135                         Tutorial tutorial = slot.
    getTutorial();
136                         AID studentOwningSlot =
    slot.getTimeslotOwner();
137                         System.out.println("
    Student: " + studentOwningSlot + " belongs \nto
    the " + tutorial.getDay() + " slot which starts at
     " + tutorial.getStartTime() + ".");
138                         timetableList.add(tutorial
    );
139 
```

```
140                         DFAgentDescription
    template = new DFAgentDescription();
141                         ServiceDescription desc =
    new ServiceDescription();
142
143                         int util = utilityFunction
    (tutorial);
144                         satisfactionLevel =
    satisfactionLevel + util;
145                         System.out.println("
    Student: " + studentOwningSlot + " satisfaction \n
    is " + util + ".");
146
147                         if (util ⩽ 0) {
148                             ACLMessage swapMsg =
    new ACLMessage(ACLMessage.CFP);
149                             desc.setType("
    TimetablingAgent");
150                             template.addServices(
    desc);
151                             try {
152                                 DFAgentDescription
    [] resultDfd = DFService.search(myAgent, template
    );
153
154                                 if (resultDfd.
    length > 0) {
155                                     swapMsg.
    addReceiver(resultDfd[0].getName());
156
157                                     swapMsg.
    setLanguage(codec.getName());
158                                     swapMsg.
    setOntology(timetableOntology.getName());
159                                     swapMsg.
    setConversationId("timetabling");
160
161                                     try {
162                                         System.out
    .println("Student " + resultDfd[0].getName() + "
    sent \ntheir slot to the timetable agent.");
163
    getContentManager().fillContent(swapMsg, slot);
164                                     send(
```

```
164 swapMsg);
165                                     } catch (
    CodecException ce2) {
166                                         ce2.
    printStackTrace();
167                                     } catch (
    OntologyException oe) {
168                                         oe.
    printStackTrace();
169                                     }
170                                 }
171                             } catch (FIPAException
     fe) {
172                                 fe.printStackTrace
    ();
173                             }
174 
175                         }
176                     }
177                 } catch (CodecException ce) {
178                     ce.printStackTrace();
179                 } catch (OntologyException oe) {
180                     oe.printStackTrace();
181                 }
182             }
183         }
184     }
185 
186     private class handleTimetablingAgentMessage
    extends CyclicBehaviour {
187         public void action() {
188             MessageTemplate mt = MessageTemplate.
    MatchPerformative(ACLMessage.INFORM);
189             ACLMessage msg = myAgent.receive(mt);
190             if (msg ≠ null) {
191                 //End of ticks
192                 if (msg.getContent().equals("
    takedownRequest")) {
193                     System.out.println(myAgent.
    getAID().getName() + " is terminating");
194                     myAgent.doDelete();
195                 } else if (msg.getContent().equals
    ("tickInform")) {
196                     //A boolean holding whether or
```

```
196  not the student has to make their swap available
197                      boolean swapWanted = false;
198
199
200                      //If result of utility is
    below 0, student needs to make their swap
    available
201                      for (int i = 0; i <
    timetableList.size(); i++) {
202                          int util = utilityFunction
    (timetableList.get(i));
203                          satisfactionLevel = 0;
204                          satisfactionLevel =
    satisfactionLevel + util;
205                          if (util < 0) {
206                              swapWanted = true;
207                          }
208                      }
209                      if (swapWanted == true) {
210                          DFAgentDescription
    template = new DFAgentDescription();
211                          ServiceDescription desc =
    new ServiceDescription();
212                          ACLMessage cfp = new
    ACLMessage(ACLMessage.REQUEST);
213                          desc.setType("
    TimetablingAgent");
214                          template.addServices(desc
    );
215                          try {
216                              DFAgentDescription[]
    result = DFService.search(myAgent, template);
217                              if (result.length > 0
    ) {
218                                  cfp.addReceiver(
    result[0].getName());
219                              }
220                          } catch (FIPAException fe
    ) {
221                              fe.printStackTrace();
222                          }
223                          cfp.setContent("
    requestSlots");
224                          cfp.setConversationId("
```

```
224 TimetablingAgent");
225                         cfp.setReplyWith("cfp" +
    System.currentTimeMillis()); // Unique value
226                         myAgent.send(cfp);
227                     }
228                 }
229             }
230         }
231     }
232
233     private class handleSwapWanted extends
    CyclicBehaviour {
234         public void action() {
235             MessageTemplate mt = MessageTemplate.
    MatchPerformative(ACLMessage.AGREE);
236             ACLMessage msg = myAgent.receive(mt);
237
238             if (msg ≠ null) {
239                 ContentElement ce = null;
240                 try {
241                     ce = getContentManager().
    extractContent(msg);
242                     if (ce instanceof
    AvailableTimeslots) {
243                         AvailableTimeslots
    availableTimeslots = (AvailableTimeslots) ce;
244
245                         int slotWanted = -1;
246                         for (int i = 0; i <
    availableTimeslots.getTimeslots().size(); i++) {
247                             for (int j = 0; j <
    timetableList.size(); j++) {
248                                 //If slots are
    compatible i.e. of the same module
249                                 if (
    availableTimeslots.getTimeslots().get(i).
    getModuleName().equals(timetableList.get(j).
    getModuleName())) {
250
251                                     int
    swappedSlotSatisfactionLevel = utilityFunction(
    availableTimeslots.getTimeslots().get(i));
252                                     int
    currentSatisfactionLevel = utilityFunction(
```

```
252 timetableList.get(j));
253                                     if (
    swappedSlotSatisfactionLevel >
    currentSatisfactionLevel) {
254                                         System.out
    .println("The current satisfaction level is " +
    currentSatisfactionLevel
255                                                 +
    ". The proposed swap would result in a level of "
     + swappedSlotSatisfactionLevel);
256                                         slotWanted
     = i;
257                                     }
258                                 }
259                             }
260                         }
261
262                         System.out.println("The
    slot wanted by the student " + myAgent.getName
     () + " (i.e. index of for loop) is slot " +
    slotWanted);
263
264                         if (slotWanted ≠ -1) {
265                             SwapProposal proposal
     = new SwapProposal();
266                             proposal.
    setTimeslotOwner(availableTimeslots.getTimeslots
    ().get(slotWanted).getAttendee());
267                             proposal.setTimeslot(
    availableTimeslots.getTimeslots().get(slotWanted
    ));
268                             proposal.
    setTimeslotReceiver(myAgent.getAID());
269
270                             ACLMessage
    requestSwapWithTimetablerMsg = new ACLMessage(
    ACLMessage.PROPOSE);
271                             DFAgentDescription
    template = new DFAgentDescription();
272                             ServiceDescription
    desc = new ServiceDescription();
273
274                             desc.setType("
    TimetablingAgent");
```

```
275                                 template.addServices(
    desc);
276                                 try {
277                                     DFAgentDescription
    [] result = DFService.search(myAgent, template);
278                                     if (result.length
     > 0) {
279 
    requestSwapWithTimetablerMsg.addReceiver(result[0
    ].getName());
280                                     }
281                                 } catch (FIPAException
     fe) {
282                                     fe.printStackTrace
    ();
283                                 }
284 
    requestSwapWithTimetablerMsg.setLanguage(codec.
    getName());
285 
    requestSwapWithTimetablerMsg.setOntology(
    timetableOntology.getName());
286 
287                                 Action request = new
    Action();
288                                 request.setAction(
    proposal);
289                                 request.setActor(
    proposal.getTimeslotReceiver());
290                                 try {
291                                     System.out.println
    ("\nStudent is requesting a swap with the
    Timetabling Agent...");
292                                     // Let JADE
    convert from Java objects to string
293                                     getContentManager
    ().fillContent(requestSwapWithTimetablerMsg,
    request);
294                                     send(
    requestSwapWithTimetablerMsg);
295                                     System.out.print("
    The swap has been sent for the Timetabling Agent
    to handle.");
296                                 } catch (
```

```
296 CodecException de) {
297                                     de.printStackTrace
    ();
298                                 } catch (
    OntologyException oe) {
299                                     oe.printStackTrace
    ();
300                                 }
301
302                             }
303                         }
304                     } catch (CodecException |
    OntologyException e) {
305                         e.printStackTrace();
306                     }
307                 }
308             }
309         }
310
311         private class updateSlot extends
    CyclicBehaviour {
312             public void action() {
313                 MessageTemplate mt = MessageTemplate.
    MatchPerformative(ACLMessage.ACCEPT_PROPOSAL);
314                 ACLMessage msg = myAgent.receive(mt);
315
316                 if (msg =≠ null) {
317                     ContentElement ce = null;
318                     try {
319                         ce = getContentManager().
    extractContent(msg);
320                         if (ce instanceof Timeslot) {
321                             Timeslot newSlot = (
    Timeslot) ce;
322                             for (int i = 0; i <
    timetableList.size(); i++) {
323                                 if (timetableList.get(
    i).getModuleName().equals((newSlot.getTutorial().
    getModuleName()))) {
324                                     timetableList.
    remove(i);
325                                     timetableList.add(
    newSlot.getTutorial());
326                                     System.out.println
```

```
326 ("Student " + myAgent.getName() + " has swapped
    for the " +
327                                             newSlot.
    getTutorial().getDay() + " " + newSlot.getTutorial
    ().getModuleName() + " slot." );
328                             }
329                         }
330                     }
331                 } catch (CodecException |
    OntologyException e) {
332                     e.printStackTrace();
333                 }
334             }
335         }
336     }
337
338     private class giveSatisfaction extends
    CyclicBehaviour {
339         public void action() {
340             MessageTemplate mt = MessageTemplate.
    MatchPerformative(ACLMessage.REQUEST);
341             ACLMessage msg = myAgent.receive(mt);
342             if (msg ≠ null) {
343                 if (msg.getContent().equals("
    satisfactionRequest")) {
344                     System.out.println("Agent " +
    myAgent.getName() + " has a satisfaction level of
     " + satisfactionLevel + ".");
345                     int totalSatisfaction = 0;
346                     for (int i = 0; i <
    timetableList.size(); i++) {
347                        totalSatisfaction =
    totalSatisfaction + satisfactionLevel;
348                     }
349                     DFAgentDescription template =
    new DFAgentDescription();
350                     ServiceDescription desc = new
    ServiceDescription();
351                     ACLMessage cfp = new
    ACLMessage(ACLMessage.PROPAGATE);
352                     desc.setType("TimetablingAgent
    ");
353                     template.addServices(desc);
354                     try {
```

```
355                         DFAgentDescription[]
    result = DFService.search(myAgent, template);
356                         if (result.length > 0) {
357                             cfp.addReceiver(result
    [0].getName());
358                         }
359                     } catch (FIPAException fe) {
360                         fe.printStackTrace();
361                     }
362                     cfp.setContent(Integer.
    toString(totalSatisfaction));
363                     cfp.setConversationId("
    TimetablingAgent");
364                     cfp.setReplyWith("cfp" +
    System.currentTimeMillis());
365                     myAgent.send(cfp);
366                 }
367             }
368         }
369     }
370
371 }
372
```

```java
package timetable_ontology;

import jade.content.AgentAction;
import jade.core.AID;

public class SwapProposal implements AgentAction {

    private AID timeslotOwner;
    private AID timeslotReceiver;
    private Tutorial timeslot;

    public AID getTimeslotOwner() {
        return timeslotOwner;
    }

    public void setTimeslotOwner(AID timeslotOwner
) {
        this.timeslotOwner = timeslotOwner;
    }

    public AID getTimeslotReceiver() {
        return timeslotReceiver;
    }

    public void setTimeslotReceiver(AID
timeslotReceiver) {
        this.timeslotReceiver = timeslotReceiver;
    }

    public Tutorial getTimeslot() {
        return timeslot;
    }

    public void setTimeslot(Tutorial timeslot) {
        this.timeslot = timeslot;
    }

}

```

```java
package timetable_ontology;

import java.util.ArrayList;
import java.util.List;
import jade.content.Predicate;

public class AvailableTimeslots implements Predicate {

    private List<Tutorial> timeslots = new ArrayList<>();

    public List<Tutorial> getTimeslots() {
        return timeslots;
    }

    public void setTimeslots(List<Tutorial> timeslots) {
        this.timeslots = timeslots;
    }

}

```

```
 1 import jade.core.*;
 2 import jade.core.Runtime;
 3 import jade.wrapper.AgentController;
 4 import jade.wrapper.ContainerController;
 5
 6
 7 public class Application {
 8     public static void main(String[] args) {
 9         ///Types of agent "Unable", "Prefer Not", "
   Neutral", "Would Like"
10         Profile myProfile = new ProfileImpl();
11         Runtime myRuntime = Runtime.instance();
12         ContainerController myContainer = myRuntime
   .createMainContainer(myProfile);
13         try{
14             AgentController rma = myContainer.
   createNewAgent("rma", "jade.tools.rma.rma", null);
15             rma.start();
16
17             StudentPreferences prefA = new
   StudentPreferences("Unable", "Tuesday", 1200, 1700
   );
18             StudentPreferences[] preferencesA = {
   prefA};
19
20             StudentPreferences prefB = new
   StudentPreferences("Unable", "Friday", 1100, 1700);
21             StudentPreferences[] preferencesB = {
   prefB};
22
23             StudentPreferences prefC = new
   StudentPreferences("Unable", "Friday", 1500, 1700);
24             StudentPreferences[] preferencesC = {
   prefB};
25
26             AgentController TimetableAgent =
   myContainer.createNewAgent("Timetabler",
   TimetableAgent.class.getCanonicalName(), null);
27             TimetableAgent.start();
28
29             AgentController StudentA = myContainer.
   createNewAgent("StudentA", StudentAgent.class.
   getCanonicalName(), preferencesA);
30             StudentA.start();
```

```
31
32              AgentController StudentB = myContainer.
   createNewAgent("StudentB", StudentAgent.class.
   getCanonicalName(), preferencesB);
33              StudentB.start();
34
35              AgentController StudentC = myContainer.
   createNewAgent("StudentC", StudentAgent.class.
   getCanonicalName(), preferencesC);
36              StudentC.start();
37
38              //AgentController StudentD =
   myContainer.createNewAgent("StudentD", StudentAgent
   .class.getCanonicalName(), preferencesC);
39              //StudentB.start();
40
41          }
42          catch(Exception e){
43              System.out.println("Exception starting
   agent: " + e.toString());
44          }
45      }
46
47      private class testOne {
48
49      }
50
51  }
52
```

```
 1
 2 public class StudentPreferences {
 3     String level;
 4     String day;
 5     int startTime;
 6     int endTime;
 7
 8     public String getLevel() {
 9         return level;
10     }
11
12     public void setLevel(String level) {
13         this.level = level;
14     }
15
16     public String getDay() {
17         return day;
18     }
19
20     public void setDay(String day) {
21         this.day = day;
22     }
23
24     public int getStartTime() {
25         return startTime;
26     }
27
28     public void setStartTime(int startTime) {
29         this.startTime = startTime;
30     }
31
32     public int getEndTime() {
33         return endTime;
34     }
35
36     public void setEndTime(int endTime) {
37         this.endTime = endTime;
38     }
39
40     public StudentPreferences(String level, String
   day, int sTime, int eTime) {
41         this.level = level; //Unable, prefer, would
    like
42         this.day = day;
```

```
43          startTime = sTime;
44          endTime = eTime;
45      }
46      public StudentPreferences() {
47
48      }
49
50
51  }
52
```

```java
package timetable_ontology;

import jade.content.onto.BeanOntology;
import jade.content.onto.BeanOntologyException;
import jade.content.onto.Ontology;

public class TimetableOntology extends BeanOntology {

    //Standard ontology variable reccommened by JADE
    private static final long serialVersionUID = 1L;

    private static Ontology theInstance = new TimetableOntology("my_ontology");

    public static Ontology getInstance(){
        return theInstance;
    }

    private TimetableOntology(String name) {
        super(name);
        try {
            add(Tutorial.class);
            add(Timeslot.class);
            add(AvailableTimeslots.class);
            add(SwapProposal.class);

        } catch (BeanOntologyException e) {
            e.printStackTrace();
        }
    }
}

```