```java
package set10111.simulation;

import java.util.ArrayList;

// was previously named as BuyerAgent because similar to buyer
public class StudentAgent extends Agent {

    // use contract net protocal or CFP to ask other students if they want to swap
    // the slot included in the request. if they do they should send details of
    // their slot back to the requesting agent
    // Run the utility function before sending CFP back to see what effect can
    // swapping have on the agent who is ready to swap.

    // Use following for the utility calculation
//              if assiginedtutorial time is in the  "would like to have"  add 10 points
//              if assiginedtutorial time is in the "outside work or caring commitments"
  subtract 10 points
//              if assiginedtutorial time is in the "prefer not to have"   subtract 2
  points

//          you can choose on the how many points to subtract or add for each agent

    // Or use the advertisement agent to see which aganet wants to swap the slots
    // and instead of sending requets directky to students to ask for slots tp swap
    // send them to the advertisement agent.
    // and when a slot to swap is found, contact the student agent directly and
    // perform the swap.

    // One of the reason why advertisement is not suitable because you will have to
    // involve an extra agent for advertisements which will also need to be updated
    // whenever a swap is completed. What i mena is
    // is that one every complete swap Timetabling agent or students linked to swap
    // will have to tell advertiser to remove or update their listings on the board

    // Use either CFP or Advertisement technique and discuss in report why you
    // choose one and why did not choose other

    private ArrayList<AID> students = new ArrayList<>();
    private AID timeTablingAgent;
    private ArrayList<String> booksToBuy = new ArrayList<>();
    private HashMap<String, TimeSlot> timeSlotsTOSwap = new HashMap<String, TimeSlot>();
    private HashMap<String, ArrayList<Offer>> currentOffers = new HashMap<>();
    private AID tickerAgent;
    private int numQueriesSent;
    ArrayList<String> moduleNames = new ArrayList<String>();
    public HashMap<String, String> myPreferences = new HashMap<String, String>();
    public HashMap<String, TimeSlot> assignedTimeSlots = new HashMap<String, TimeSlot>();
    public HashMap<String, ContentElementList> activeProposals = new HashMap<String,
  ContentElementList>();
    private int numSwapRequestsSent = 0;

    private Codec codec = new SLCodec();
    private Ontology ontology = TimetablingOntology.getInstance();

    @Override
    protected void setup() {

        // add this agent to the yellow pages
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(getAID());
        ServiceDescription sd = new ServiceDescription();
        sd.setType("student");
```

```
95          sd.setName(getLocalName() + "-student-agent");
96          dfd.addServices(sd);
97          try {
98              DFService.register(this, dfd);
99          } catch (FIPAException e) {
100             e.printStackTrace();
101         }
102
103         getContentManager().registerLanguage(codec);
104         getContentManager().registerOntology(ontology);
105
106         // add books to buy
107         // get the title of book from passed arguments to agent
108         Object[] args = getArguments();
109         HashMap<String, String> myreferences= ((Customm) args[0]).preferences;
110         myPreferences = myreferences;
111
112         SequentialBehaviour dailyActivity = new SequentialBehaviour();
113         // sub-behaviours will execute in the order they are added
114         dailyActivity.addSubBehaviour(new FindSudents(this));
115         dailyActivity.addSubBehaviour(new PreferenceGenerator(this));
116         dailyActivity.addSubBehaviour(new GetTimetableBehaviour());
117         // send Enquiries will only be called if above behaviour is complete. But we may
118         // need it to update the timetables if slots are swapped
119         // that means the send Enquiries will never be called OR make some different
120         // kind of behaviour
121         // add it to the setup()
122         dailyActivity.addSubBehaviour(new SendEnquiries(this));
123         dailyActivity.addSubBehaviour(new CollectOffers(this));
124         addBehaviour(dailyActivity);
125         addBehaviour(new QueryBehaviour());
126
127         addBehaviour(new ProposalResponseServer());
128     }
129
130     @Override
131     protected void takeDown() {
132         // Deregister from the yellow pages
133         try {
134             DFService.deregister(this);
135         } catch (FIPAException e) {
136             e.printStackTrace();
137         }
138     }
139
140     public class PreferenceGenerator extends OneShotBehaviour {
141
142         public PreferenceGenerator(Agent a) {
143             super(a);
144         }
145
146         @Override
147         public void action() {
148             try {
149
150                 // only commented for testing
151
152 //              ArrayList<String> prefCodes = new ArrayList<String>();
153 //              prefCodes.add("outside work or caring commitments");
154 //              prefCodes.add("prefer not to attend");
155 //              prefCodes.add("like to have");
156 //
```

```
157 //              // add new dummy module
158 ////                moduleNames.add("Module1");
159 ////                moduleNames.add("Module2");
160 ////                moduleNames.add("Module3");
161 //
162 //              // commented below for because we do not need to have preferences separatly
    for
163 //              // ech module. they are for common for all modules
164 ////                for (String moduleName : moduleNames) {
165 ////
166 ////                    Module module = new Module();
167 //
168 //              for (int i = 0; i < 5; i++) { // add preferences for 5 days
169 //
170 //                  int day = i + 1;
171 //
172 //                  // assign time slots for each day for hours between 9-17
173 //                  for (int j = 9; j < 18; j++) {
174 //
175 //                      int random = (int) Math.round((0 + 2 * Math.random())); // generate
    random number between 0
176 //                                                                              // and 2
177 //                      int startTime = j;
178 //                      int endTime = j + 1;
179 //
180 //                      String preferenceCode = prefCodes.get(random); // get random
    preference
181 //
182 //                      TimeSlot slot = new TimeSlot();
183 //                      slot.setModuleName("");
184 //                      slot.setGroupId(9); // should be optional because enot needed for
    preferences
185 //                      slot.setDate(day);
186 //                      slot.setStartTime(startTime);
187 //                      slot.setEndTime(endTime);
188 //                      slot.setStatus(preferenceCode);
189 //
190 //                      String slotID =
    String.valueOf(day).concat(String.valueOf(startTime))
191 //                              .concat(String.valueOf(endTime));
192 //
193 //                      myPreferences.put(slotID, slot.getStatus());
194 //
195 //                      System.out.println(getLocalName() + "   " + " -------  " +
    slot.getModuleName() + " "
196 //                              + slot.getGroupId() + " " + i + " " + slot.getStartTime() +
    " " + slot.getEndTime()
197 //                              + "  " + preferenceCode);
198 //                  }
199 //
200 ////                        preferences.moduleList.put(moduleName, module);
201 //
202 //              }
203
204                 int total = utilityFunction(assignedTimeSlots, myPreferences);
205
206                 System.out.println("Total utility  : " + total);
207 //              }
208
209             } catch (Exception e) {
210                 e.printStackTrace();
211             }
```

```
212
213         }
214     }
215
216     public class FindSudents extends OneShotBehaviour {
217
218         public FindSudents(Agent a) {
219             super(a);
220         }
221
222         @Override
223         public void action() {
224             DFAgentDescription sellerTemplate = new DFAgentDescription();
225             ServiceDescription sd = new ServiceDescription();
226             sd.setType("student");
227             sellerTemplate.addServices(sd);
228             try {
229                 students.clear();
230                 DFAgentDescription[] agentsType1 = DFService.search(myAgent,
    sellerTemplate);
231                 for (int i = 0; i < agentsType1.length; i++) {
232                     // check if the aid of the student is not the AID of this student. If
    it is then
233                     // do not add to other student agents list
234                     if
    (!agentsType1[i].getName().getLocalName().contentEquals(myAgent.getAID().getLocalName())) {
235                         students.add(agentsType1[i].getName()); // this is the AID
236                     }
237                 }
238                 System.out.println("Total students : " + students.size());
239             } catch (FIPAException e) {
240                 e.printStackTrace();
241             }
242
243         }
244
245     }
246
247     private class GetTimetableBehaviour extends Behaviour {
248
249         @Override
250         public boolean done() {
251             // TODO Auto-generated method stub
252             if (timeSlotsTOSwap.size() != 0) {
253                 return true;
254             }
255             return false;
256         }
257
258         @Override
259         public void action() {
260             // This behaviour should only respond to REQUEST messages
261             MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.INFORM); //
    chnage to request
262             ACLMessage msg = receive(mt);
263             if (msg != null) {
264                 try {
265                     ACLMessage reply = msg.createReply();
266                     ContentElement ce = null;
267                     System.out.println(msg.getContent()); // print out the message content
    in SL
268
```

```
269                     // Let JADE convert from String to Java objects
270                     // Output will be a ContentElement
271                     ce = getContentManager().extractContent(msg);
272                     if (ce instanceof Action) {
273                         Concept action = ((Action) ce).getAction();
274                         if (action instanceof AcceptTimetable) {
275                             AcceptTimetable order = (AcceptTimetable) action;
276                             TimeTable it = order.getTimeTable();
277                             timeTablingAgent = order.getSenderAgentAid(); // get the aid of timetabling agent which will
278                                                                           // be used whenever a timeslot is needed to
279                                                                           // swap. It will be then used to to ask
280                                                                           // Timetabling agent to perform swapping
281 //                          System.out.println(timeTablingAgent);
282                             // Extract the CD name and print it to demonstrate use of the ontology
283                             if (it instanceof TimeTable) {
284                                 TimeTable tm = (TimeTable) it;
285 
286                                 // store time table
287                                 Random random = new Random();
288                                 for (int i = 0; i < tm.getTutorialAssignment().size(); i++) {
289 
290                                     TimeSlot slot = tm.getTutorialAssignment().get(i);
291                                     String slotID = String.valueOf(slot.getDate()).concat(String.valueOf(slot.getStartTime()))
292                                             .concat(String.valueOf(slot.getEndTime()));
293                                     assignedTimeSlots.put(slotID, slot); // adding slot to tutorial times arraylist
294                                                                          // assignedTimeSlots
295 
296                                 }
297 
298                                 // calculating utility of the assigned time table using preferences
299                                 int total = utilityFunction(assignedTimeSlots, myPreferences);
300 
301                                 System.out.println("Total utility  : " + total);
302 
303                                 System.out.println("Total time slots to swap are : " + timeSlotsTOSwap.size());
304 
305                                 if (timeSlotsTOSwap.size() < 1) {
306                                     myAgent.doDelete();
307                                 }
308                             }
309                         }
310 
311                     }
312                 }
313 
314                 catch (CodecException ce) {
315                     ce.printStackTrace();
316                 } catch (OntologyException oe) {
317                     oe.printStackTrace();
318                 }
319 
```

```
320             } else {
321                 block();
322             }
323         }
324     }
325 
326 
327     // shift these two functions in helper class
328     public int utilityFunction(HashMap<String, TimeSlot> assignedTimeSlots,
329             HashMap<String, String> studentPreferences) {
330         int totalFitness = 0;
331         for (String slotID : assignedTimeSlots.keySet()) {
332 //          String slotID =
    String.valueOf(slot.getDate()).concat(String.valueOf(slot.getStartTime()))
333 //                  .concat(String.valueOf(slot.getEndTime()));
334 
335             String studentPreference = studentPreferences.get(slotID);
336 
337             TimeSlot slot = assignedTimeSlots.get(slotID);
338 
339             int chnageInFitness = this.calculateFitnessChange(studentPreference);
340 
341             if (chnageInFitness == 8 || chnageInFitness == -10) {
342                 timeSlotsTOSwap.put(slotID, slot);
343             }
344             totalFitness = totalFitness + chnageInFitness;
345         }
346         return totalFitness;
347     }
348 
349     public int calculateFitnessChange(String studentPreference) {
350         // total points should be 30 if all modules time slots match students
351         // preferemces.
352         // If time slot does not match preference than substract 10 points
353         // if amtch then add 10 points
354         // if timeslot timetable is in "prefer not to attend" then only give 8 points
355         // which is 2 points less than if it was "liketo have"
356         int fitness = 0;
357 //
358 //      studentPreference = myPreferences.get(slotID);
359 
360         if (studentPreference == "like to have") {
361 
362             fitness += 10; // add 10 pints to fitness if student-preference time matches
    with slot time
363         } else if (studentPreference == "prefer not to attend") {
364 
365             fitness = fitness + 8; // add 8 points to fitness if student-preference at the
    time of slot's time was
366                                    // "prefer not to attend"
367         } else if (studentPreference == "outside work or caring commitments") {
368 
369             fitness = fitness - 10; // subtract 10 points from fitness if
    student-preference at the time of slot's
370                                    // time was "outside work or caring commitments"
371         }
372         return fitness;
373     }
374 
375     private TimeSlot slotWithWorstFitness(HashMap<String, TimeSlot> slots, String
    moduleName) {
376         String worstfitKey=null;
```

```
377         int oldFitness=10;
378         TimeSlot slot = new TimeSlot();
379         for (String id : slots.keySet()) {
380             slot = slots.get(id);
381             String pref = myPreferences.get(id);
382             int fitness = calculateFitnessChange(pref);
383             if (fitness <= oldFitness && slot.getModuleName().contains(moduleName)) { //&&
    slot.getModuleName() == cfpSlot.getModuleName() && slot.getGroupId() !=
    cfpSlot.getGroupId()
384                 worstfitKey = id;
385             }
386             oldFitness = fitness;
387         }
388         return slots.get(worstfitKey);
389     }
390 
391 
392     public class SendEnquiries extends OneShotBehaviour {
393 
394         public SendEnquiries(Agent a) {
395             super(a);
396         }
397 
398         @Override
399         public void action() {
400             // send out a call for proposals for each book
401             numQueriesSent = 0;
402             System.out.println(myAgent.getLocalName());
403             for (String string : assignedTimeSlots.keySet()) {
404                 System.out.println(string);
405             }
406             int numQueriesSentForEachSlot = 0;
407             for (String slotID : timeSlotsTOSwap.keySet()) {
408 
409                 // Prepare the Query-IF message
410                 ACLMessage enquiry = new ACLMessage(ACLMessage.CFP); // change to CFP
411                 enquiry.setLanguage(codec.getName());
412                 enquiry.setOntology(ontology.getName());
413 
414                 for (AID student : students) {
415                     // Prepare the content using predicate.
416                     TimeSlot tutorialSlot = timeSlotsTOSwap.get(slotID);
417 
418                     enquiry.setConversationId(slotID + myAgent.getName());
419                     Swap swapSlot = new Swap(); // rename the OwnsTimeSlot
420                     swapSlot.setItem(tutorialSlot);
421                     swapSlot.setOwner(student);
422 
423                     enquiry.addReceiver(student); // student is the AID of the student
    agent to whom the enquiry is sent
424 
425                     // IMPORTANT: According to FIPA, we need to create a wrapper Action
    object
426                     // with the action and the AID of the agent
427                     // we are requesting to perform the action
428                     // you will get an exception if you try to send the sell action
    directly
429                     // not inside the wrapper!!!
430                     Action request = new Action();
431                     request.setAction(swapSlot);
432                     request.setActor(student); // the agent that you request to perform the
    action
```

```
433
434                     try {
435                         // Let JADE convert from Java objects to string
436                         getContentManager().fillContent(enquiry, request);
437                         myAgent.send(enquiry); // send enquiry for current tutorial in loop
   to all other students
438                         numQueriesSent++; // this stores total queries sent for all
   timeslots to swap
439                         numQueriesSentForEachSlot++; // this stores total queries sent for
   for current timeslot in loop
440                     } catch (CodecException ce) {
441                         ce.printStackTrace();
442                     } catch (OntologyException oe) {
443                         oe.printStackTrace();
444                     }
445                 }
446                 System.out.println("Total enquiries sent : " + numQueriesSentForEachSlot);
447                 numQueriesSentForEachSlot = 0;
448             }
449
450         }
451     }
452
453
454     private class QueryBehaviour extends CyclicBehaviour {
455         @Override
456         public void action() {
457             // This behaviour should only respond to QUERY_IF messages
458             MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
459             ACLMessage msg = receive(mt);
460             if (msg != null) {
461                 try {
462                     ACLMessage reply = msg.createReply();
463                     ContentElement ce = null;
464 //                  System.out.println(msg.getContent()); //print out the message content
   in SL
465
466                     // Let JADE convert from String to Java objects
467                     // Output will be a ContentElement
468                     ce = getContentManager().extractContent(msg);
469
470                     // check if content is an Action
471                     if (ce instanceof Action) {
472                         Action swapAction = ((Action) ce);
473                         Concept action = swapAction.getAction();
474
475                         // check if action is of type OwnsTimeSlot
476                         if (action instanceof Swap) {
477                             Swap owns = (Swap) action;
478                             TimeSlot it = owns.getItem();
479
480                             // Extract the TimeSlot and build its id and then and print it
   to ensure message
481                             // was received successfully
482                             TimeSlot slot = (TimeSlot) it;
483                             String slotID =
   String.valueOf(slot.getDate()).concat(String.valueOf(slot.getStartTime()))
484                                     .concat(String.valueOf(slot.getEndTime()));
485
486                             System.out.println("The timeslot id is " + slotID);
487                             // check if student has this timeslot in the timeSlotsToswap
   list or check if
```

```
488                             // student wants to swap the time slot
489                             // only swap if group ids of both time slots are not same
    because no point
490                             // swapping with student from same group
491                             // todo issue - - not seen scenario so far where students have
    tutorial with
492                             // same time but different group ids.May be randomly generated
    timetable always
493                             // skip that
494                             // todo solution - - may be start passing preferences and
    timetable in main
495                             // which will allow manual creation of desired swapping
    scenarios // this checks
496                             // if student agent has not already agrred to swap this slot
    with other agent
497
498 //                          check if timeslot is not already in assignedSlots, if it is
    then no point swaping it because will make not improvement to the fitness
499                             if (!timeSlotsTOSwap.containsKey(slotID)
500                                     && !currentOffers.containsKey(slotID) &&
    timeSlotsTOSwap.size() > 0) {
501
502                                 // to=doadd this slotid to nottoswap list so that it is
    blocked from swapping because already negotiating with one student
503
504                                 // send confirm message
505
506                                 // check if the timeslot in the CFP from other student
    benefit your local fitness Or is it also not in the "prefer not to attend" and "Other
    caring time"
507
508                                 //get the timeslot with worst fitness from timeSlotsToswap
509                                 TimeSlot myWorstSlot =
    slotWithWorstFitness(timeSlotsTOSwap, slot.getModuleName());
510
511                                 // check if the time slots belong to same module
512 //                              if(myWorstSlot.getModuleName().contains(slot.getModuleName(
    ))) {
513                                 if(myWorstSlot != null) {
514                                     String worstSlotId =
    String.valueOf(myWorstSlot.getDate()).concat(String.valueOf(myWorstSlot.getStartTime()))
515                                             .concat(String.valueOf(myWorstSlot.getEndTime()
    ));
516                                     String pref = myPreferences.get(worstSlotId);
517
518                                     // store the fitness of the time slot which this
    student has on the timeSlot to swap list
519                                     int fitnessOfMySlot= myWorstSlot == null ? 20
    :calculateFitnessChange(pref);
520
521                                     pref = myPreferences.get(slotID);
522                                     // fitness of the timeslot which other student agent is
    asking proposal for
523                                     int fitnessofCFPslot = calculateFitnessChange(pref);
524
525                                     // check if CFP timeslot will benefit this student
    agent
526                                     if (fitnessofCFPslot > fitnessOfMySlot) {
527
528                                         reply.setPerformative(ACLMessage.PROPOSE);
529
530                                         // craete content element list which will have
```

```
orignal action requested by other
                                        // student and offer of this student
                                        ContentElementList cml = new ContentElementList();
                                        cml.add(swapAction);

                                        // craete response offer
                                        AgreeSwap offer = new AgreeSwap();
                                        // add the timeslot this student want to swap for
the timeslot which came in CFP
                                        offer.setItem(myWorstSlot);
                                        offer.setResponse(true);

                                        // add offer to the content list a swell
                                        cml.add(offer);

                                        // Let JADE convert from Java objects to string
                                        getContentManager().fillContent(reply, cml);
                                        System.out.println("-----" + cml);
                                        myAgent.send(reply); // send reply/offer to other
students

                                        // add on the list consersation id to keep record
of the proposal (cml which contain myworst and CFPslot) which has been sent to another
student agent
                                        activeProposals.put(msg.getConversationId(), cml);
//                                      System.out.println("I have the time slot and want
to swap!");
                                    }
                                    else {
//                                      System.out.println("Your slot not increasing my
fitness");

                                        reply.setPerformative(ACLMessage.REFUSE);
                                        send(reply);
                                    }
                                }
                                else {
//                                  System.out.println("Module mismatch");
                                    reply.setPerformative(ACLMessage.REFUSE);
                                    send(reply);
                                }

                            } else {
//                              System.out.println("I dont want to swap time slot");

                                reply.setPerformative(ACLMessage.REFUSE);
                                send(reply);
                                if (timeSlotsTOSwap.size() < 1) {
                                    myAgent.doDelete();
                                }
                            }
                        }
                    }

                }

                catch (CodecException ce) {
                    ce.printStackTrace();
                } catch (OntologyException oe) {
                    oe.printStackTrace();
                }

            } else { // if no que in the message que block the behaviour
```

```
586                 block();
587             }
588         }
589 
590     }
591 
592     public class CollectOffers extends Behaviour {
593         private int numRepliesReceived = 0;
594         boolean received = false;
595         private int step = 0;
596         private ACLMessage replyProposal;
597 
598         public CollectOffers(Agent a) {
599             super(a);
600             currentOffers.clear();
601         }
602 
603         @Override
604         public void action() {
605             switch (step) {
606             case 0:
607                 for (String slotID : timeSlotsTOSwap.keySet()) {
608                     MessageTemplate mt = MessageTemplate.MatchConversationId(slotID +
    myAgent.getName());
609                     ACLMessage msg = myAgent.receive(mt);
610                     if (msg != null) {
611                         received = true;
612                         numRepliesReceived++;
613                         replyProposal = msg.createReply();
614                         if (msg.getPerformative() == ACLMessage.PROPOSE) {
615 //                          System.out.println("Propose received");
616                             ContentElement cel = null;
617                             try {
618                                 cel = getContentManager().extractContent(msg);
619 
620                                 ContentElementList cl = ((ContentElementList) cel);
621                                 ContentElement ceFromCml = cl.get(1);   // usong one
    because offer/AgreeSwap is stored at index 1
622                                 System.out.println(ceFromCml);
623 
624                                 if (ceFromCml instanceof AgreeSwap) {
625                                     AgreeSwap resp = (AgreeSwap) ceFromCml;
626                                     TimeSlot propSlot = resp.getItem();
627 //                                  System.out.println("The offer received " + resp);
628 
629                                     String propSlotId =
    String.valueOf(propSlot.getDate()).concat(String.valueOf(propSlot.getStartTime()))
630                                             .concat(String.valueOf(propSlot.getEndTime()));
631 
632                                     // check propSlot is already not in my
    assignedTimetable
633                                     if(!timeSlotsTOSwap.containsKey(propSlotId)) {
634 
635                                         // calculate utility to see if it inacreses utility
    points or not
636                                         //get fitness of the timeslot which was sent to
    other student as CFP
637                                         TimeSlot mySlot = timeSlotsTOSwap.get(slotID);
638                                         String pref = myPreferences.get(slotID);
639                                         // store the fitness of the time slot which this
    student sent to other student as CFPhas on the timeSlot to swap list
640                                         int fitnessOfMySlot= calculateFitnessChange(pref);
```

```
641                                     // fitness of the timeslot which other student
   agent has replied back with
642
643                                     pref = myPreferences.get(propSlotId);
644                                     int fitnessofPropslot =
   calculateFitnessChange(pref);
645
646                                     // check if propSlot timeslot will benefit this
   student agent
647                                     //
648                                     if (fitnessofPropslot >= fitnessOfMySlot) {
649                                         // add offer to the current offers hashmap list
   when offer for that slot id is
650                                         // recived first time
651                                         if (!currentOffers.containsKey(slotID)) {
652                                             ArrayList<Offer> offers = new
   ArrayList<>();
653                                             offers.add(new Offer(msg.getSender(),
   propSlot));
654                                             currentOffers.put(slotID, offers);
655                                         }
656                                         // otherwise add subsequent offers to the
   existing current offers list
657                                         else {
658                                             ArrayList<Offer> offers =
   currentOffers.get(slotID);
659                                             offers.add(new Offer(msg.getSender(),
   propSlot));
660                                         }
661                                     }
662                                     else {
663                                         replyProposal.setPerformative(ACLMessage.REJECT
   _PROPOSAL);
664                                         myAgent.send(replyProposal);
665                                     }
666                                 }else {
667                                     replyProposal.setPerformative(ACLMessage.REJECT_PRO
   POSAL);
668                                     myAgent.send(replyProposal);
669                                 }
670                             }
671
672                         } catch (CodecException ce) {
673                             ce.printStackTrace();
674                         } catch (OntologyException oe) {
675                             oe.printStackTrace();
676                         }
677                     } else if (msg.getPerformative() == ACLMessage.REFUSE) { // &
   numRepliesReceived ==
678
   // numQueriesSent
679                         // if we have received all the replies that means no other
   student want to swap
680                         // the slot.
681                         // So we should not send anymore CFP's for this slot and it
   should be removed
682                         // from the slotsToSwap array
683                         System.out.println("Refuse received");
684                     }
685                 }
686             }
687
```

```
688                 if (numQueriesSent == numRepliesReceived) {
689                     step =1;
690                     break;
691                 }
692 
693             case 1:         // remove slots which didnot get any offers
694                 Set<String> ids = timeSlotsTOSwap.keySet();
695                 ArrayList<String> notToSwap = new ArrayList<String>();
696                 boolean sendrequst = false;
697                 if (numQueriesSent == numRepliesReceived && currentOffers.size() > 0) {
698                     for (String id : currentOffers.keySet()) {
699                         for (String slotToSwapId : ids) {
700                             if (currentOffers.containsKey(slotToSwapId) ) {
701                                 // if received offer for slotToSwapId then send request to
    Timerabling agent to perform swap
702 //                                addBehaviour(new SendSwapAction(myAgent)); // not needed
    here moved to line 511
703                                 sendrequst = true;
704 
705                             } else {
706                                 // otherwise that means we received no offers for
    slotToSwapId in // timeSlotsToSwap hashmap
707                                 // so remove it from the list because all other students
    have confirmed that hey do not want to swap
708                                 boolean removed = true;
709                                 notToSwap.add(slotToSwapId);
710 
711 //                                if (removed) {
712 //                                    System.out.println("Time slot is removed from the
    swapping list. You have to attend it now. "+ removed);
713 //                                }
714                             }
715                         }
716 //                        System.out.println("Total offer for " + id + " - " +
    currentOffers.get(id).size());
717 
718                     }
719 
720                     // remove timeslots which not required to swap anymore
721                     for (String id : notToSwap) {
722                         timeSlotsTOSwap.remove(id);
723                     }
724 
725                     // call one shot behaviour to send swap requets
726                     if(sendrequst) {
727                             // remove offers related to mySlotId because no longer need to
    swap that time slot and send Reject proposal to agents whose offer was not accepted
728                          TimeSlot otherSlot = new TimeSlot();
729                         for (String id : currentOffers.keySet()) {
730                             Offer bestOffer=null;
731                             int BestIndex=0;
732                             int oldFitness=0;
733                             String pref = myPreferences.get(id);
734                             oldFitness = calculateFitnessChange(pref);
735                             int i =0;
736                             for (Offer otherOffer : currentOffers.get(id)) {
737                                 otherSlot = otherOffer.getSlot();
738                                 String OfferId =
    String.valueOf(otherSlot.getDate()).concat(String.valueOf(otherSlot.getStartTime()))
739                                                     .concat(String.valueOf(otherSlot.getEndTime()))
    ;
740                                  pref = myPreferences.get(OfferId);
```

```
741                                 int fitness = calculateFitnessChange(pref);
742                                 if (fitness > oldFitness ) { //&& slot.getModuleName() ==
    cfpSlot.getModuleName() && slot.getGroupId() != cfpSlot.getGroupId()
743                                     oldFitness = fitness;
744                                     BestIndex = i;
745                                 }
746                                 i++;
747                             }
748                             bestOffer = currentOffers.get(id).remove(BestIndex);
749 
750                             // send accept proposal
751                             AID otherStudent = bestOffer.getOfferedByAID();
752                             ACLMessage propReply = new
    ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
753                             propReply.setLanguage(codec.getName());
754                             propReply.setOntology(ontology.getName());
755                             propReply.setConversationId(id + myAgent.getName());
756                             propReply.addReceiver(otherStudent);
757                             myAgent.send(propReply);
758 
759                             // if theer are anymore offers then send them reject message
760                             if (currentOffers.get(id).size() > 0) {
761                                 // reject the oher offers
762                                 for(i=0; i< currentOffers.size(); i++) {
763                                     Offer rejectOffer = currentOffers.get(id).remove(i);
764                                     // Prepare the Reject_proposal message
765                                     otherStudent = rejectOffer.getOfferedByAID();
766                                     propReply.setPerformative(ACLMessage.REJECT_PROPOSAL);
767                                     propReply.setLanguage(codec.getName());
768                                     propReply.setOntology(ontology.getName());
769                                     propReply.setConversationId(id + myAgent.getName());
770                                     propReply.addReceiver(otherStudent);
771                                     myAgent.send(propReply);
772                                 }
773                             }
774                             // add back the best offer to the list so that this agent can
    make request to timetabling agent for it
775                             currentOffers.get(id).add(bestOffer);
776                         }
777                         addBehaviour(new SendSwapAction(myAgent));
778                         step = 2;
779                     }
780                     // it is set to 2 because we have sent all the swap requests or have
    removed the slots which have not recived any offers from timeSlotsToSwap
781                     received = false;
782                 }
783 
784                 if (numQueriesSent == numRepliesReceived && (currentOffers.size() == 0 &&
    activeProposals.size() ==0)) {   // means no offers received for any time slots then
    display final fitness
785                     int total = utilityFunction(assignedTimeSlots, myPreferences);
786                     System.out.println("Final utility  of  - " + myAgent.getLocalName() + "
    : "+ total);
787                     // delete agent because all other agents refused to swap any of its
    slots
788                     myAgent.doDelete();
789                 }
790                 break;
791             case 2:
792                 for (String mySlotId : currentOffers.keySet()) {
793                     // build template to receive result from TA agent for each swapping
    request using the unique conv id
```

```
794                     String convId = mySlotId + myAgent.getName() +
    currentOffers.get(mySlotId).get(0).getOfferedByAID();
795                     AID otherStudent =
    currentOffers.get(mySlotId).get(0).getOfferedByAID();
796                     MessageTemplate mt = MessageTemplate.MatchConversationId(convId);
797
798                     // checks if TA has replied for that id
799                     ACLMessage msg = myAgent.receive(mt);
800                     if (msg != null) {
801                         received = true;
802                         if(msg.getPerformative() == ACLMessage.INFORM) {
803                             try {
804                                 ContentElement ce = null;
805                                 System.out.println(msg.getContent()); //print out the
    message content in SL
806                                 // Let JADE convert from String to Java objects // Output
    will be a ContentElement
807                                 ce = getContentManager().extractContent(msg);
808
809                                 // check if content is an Action
810                                 if (ce instanceof Result) {
811                                     Result result = ((Result) ce);
812                                     Concept action = result.getAction();
813
814                                     if(action instanceof Action) {
815                                         Concept act = ((Action) action).getAction();
816                                         // check if action is of type OwnsTimeSlot
817                                         if (act instanceof UpdateTimetable) {
818                                             UpdateTimetable owns = (UpdateTimetable) act;
819                                             TimeSlot propSlot = owns.getTimeSlot();
820
821                                             currentOffers.get(mySlotId).remove(0);
    // removed the first agent whose offer was accepted and who has already been set
    acceptproposal
822                                         // update local timetable // remove old slot
823                                             assignedTimeSlots.remove(mySlotId);
824
825                                             // create slot id for prop slot which has now
    become my new slot id there we should also have new slotid instead of old
826                                             String swappedSlotId =
    String.valueOf(propSlot.getDate()).concat(String.valueOf(propSlot.getStartTime()))
827                                                     .concat(String.valueOf(propSlot.getEndT
    ime()));
828
829                                             assignedTimeSlots.put(swappedSlotId, propSlot);
830                                             // remove the slot from timeSlots to swap
    because already swapped it
831                                             timeSlotsTOSwap.remove(mySlotId);
832
833                                             if(timeSlotsTOSwap.size() == 0 &&
    assignedTimeSlots.size() > 0) {    // means all swapped are done
834                                                 int total =
    utilityFunction(assignedTimeSlots, myPreferences);
835
836                                                 System.out.println("Final utility  of  - "
    + myAgent.getLocalName() + " : "+ total);
837
838                                                 // can delete this agent too because no
    more time slots to swap
839                                                 System.out.println(myAgent.getLocalName());
840                                                 for (String string :
    assignedTimeSlots.keySet()) {
```

```
841                                                 System.out.println(string);
842                                             }
843                                             myAgent.doDelete();
844
845                                         }
846                                     }
847                                 }
848
849                                 }
850                             } catch (CodecException ce) {
851                                 ce.printStackTrace();
852                             } catch (OntologyException oe) {
853                                 oe.printStackTrace();
854                             }
855                         }
856                         else if(msg.getPerformative() == ACLMessage.FAILURE){
857                             // send reply to other student agent that you reject his
    proposal
858                             ACLMessage proposeReply = new
    ACLMessage(ACLMessage.REJECT_PROPOSAL);
859                             proposeReply.setLanguage(codec.getName());
860                             proposeReply.setOntology(ontology.getName());
861                             proposeReply.setConversationId(mySlotId + myAgent.getName());
862                             System.out.println("Error swapping slot. Try again");
863                             proposeReply.addReceiver(otherStudent);
864                             myAgent.send(proposeReply); // sending reply to other student
    not the timetabling agent
865                         }
866                         step = 3; // case finished so assign value 3 to end behaviour
867                     }
868
869                 }
870                 break;
871             }
872             if (!received) {
873                 block();
874             }
875         }
876
877         @Override
878         public boolean done() {
879             return numRepliesReceived == numQueriesSent && step ==3;
880         }
881
882         @Override
883         public int onEnd() {
884             // print the offers
885 //          for (String slotId : timeSlotsTOSwap.keySet()) {
886 //              if (currentOffers.containsKey(slotId)) {
887 //                  ArrayList<Offer> offers = currentOffers.get(slotId);
888 //                  for (Offer o : offers) {
889 //                      System.out.println(slotId + "," + o.getSeller().getLocalName() +
    "," + o.getPrice());
890 //                  }
891 //              } else {
892 //                  System.out.println("No offers for " + slotId);
893 //              }
894 //          }
895             return 0;
896         }
897
898     }
```

```
899
900
901
902
903     public class SendSwapAction extends OneShotBehaviour {
904
905         public SendSwapAction(Agent a) {
906             super(a);
907         }
908
909         @Override
910         public void action() {
911             // send out a call for proposals for each book
912             numSwapRequestsSent = 0;
913
914             for (String slotID : currentOffers.keySet()) {
915
916                 // Prepare the REQUEST message
917                 ACLMessage swapRequest = new ACLMessage(ACLMessage.REQUEST);
918                 swapRequest.setLanguage(codec.getName());
919                 swapRequest.setOntology(ontology.getName());
920
921                 // get the first first offer from the offerlist. all other have already
922                 //been rejected in case 1 of CollectOffers behaviour
923                 Offer offer = currentOffers.get(slotID).get(0);  // Offer class move to
    ontology predicate
924
925                 // set the id of the request
926                 swapRequest.setConversationId(slotID + myAgent.getName() +
    offer.getOfferedByAID());
927
928                 UpdateTimetable swapSlot = new UpdateTimetable();
929                 swapSlot.setStudentRequested(myAgent.getAID());
930                 swapSlot.setStudentOffered(offer.getOfferedByAID());
931                 TimeSlot s = new TimeSlot();
932                 s = offer.getSlot();
933                 swapSlot.setTimeSlot(s);
934                 swapRequest.addReceiver(timeTablingAgent); // timeTablingAgent is the AID
    of the timeTablingAgent agent
935                                                            // to whom the swap request is
    sent
936
937                 // IMPORTANT: According to FIPA, we need to create a wrapper Action object
938                 // with the action and the AID of the agent
939                 // we are requesting to perform the action
940                 // you will get an exception if you try to send the sell action directly
941                 // not inside the wrapper!!!
942                 Action request = new Action();
943                 request.setAction(swapSlot);
944                 request.setActor(timeTablingAgent); // the agent that you request to
    perform the action
945
946                 try {
947                     // Let JADE convert from Java objects to string
948                     getContentManager().fillContent(swapRequest, request);
949
950                     myAgent.send(swapRequest); // send request
951                     System.out.println(swapRequest);
952                     numSwapRequestsSent++; // this stores total queries sent for all
    timeslots to swap
953                 } catch (CodecException ce) {
954                     ce.printStackTrace();
```

```
955                 } catch (OntologyException oe) {
956                     oe.printStackTrace();
957                 }
958 
959             }
960             System.out.println("Total swap requests sent to timetabling agent : " +
    numSwapRequestsSent);
961 
962         }
963     }
964 
965 
966 
967 
968     private class ProposalResponseServer extends CyclicBehaviour {
969 
970         @Override
971         public void action() {
972             // mesage template using OR to check if we have either type of message in the
    message queue
973             MessageTemplate mt =
    MessageTemplate.or(MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROPOSAL),
974                     MessageTemplate.MatchPerformative(ACLMessage.REJECT_PROPOSAL));
975             ACLMessage msg = receive(mt);
976             if (msg != null) {
977 
978                 String convId = msg.getConversationId();
979 //                try {
980                 // deals with both rejects from other student if that sudent rejects the
    proposal does not want to swap or is rejecting because that student recieved Failure from
    ta agent for swap request
981                     if (msg.getPerformative() == ACLMessage.REJECT_PROPOSAL ) {
982                         // remove the proposal from active proposal list because we have
    reecived reply related to that proposal
983                         activeProposals.remove(convId);
984 //                        System.out.println("Propose rejected by cfp student. Removed
    propose from blocking list");
985                     }
986                     else if(msg.getPerformative() == ACLMessage.ACCEPT_PROPOSAL) {
987                         // remove the proposal from active proposal list because we have
    received reply related to that proposal
988                         // also remove that slot from timeSlotsToSwap list because
    Accept_proposal means we have successfully swapped it with other student
989                         System.out.println("Proposal accepted by cfp student.");
990                         activeProposals.remove(convId);
991                     }
992             } else { // if no message in the message que block the behaviour
993                 block();
994             }
995         }
996 
997     }
998 
999 }
```