# IN4010 Practical assignment: Automated Negotiation

September 5, 2017

## 1 Introduction

Imagine you have just graduated and plan to throw a nice party for that occasion. The problem is that you do not want to organize everything yourself. Fortunately, two friends of yours have also graduated and want to throw a party as well. You decide to organize the party together, however everyone has different preferences. Being a computer science student you use your skills to make a negotiating agent so the party is as close to *your* preferences as possible.

Figure 1: Organizing a party involves choosing music, food, drinks, catering, etc.

This document describes the practical assignment for the first quarter of the AI course. The practical assignment aims at familiarizing you with parts of the course material in a more practical way. This assignment is about multilateral negotiation – a form of interaction in which more than two agents, with conflicting interests and a desire to cooperate, try to reach a mutually acceptable agreement. This assignment concerns building your own negotiating software agent. This will be a team effort: as a team **you will design, implement and analyze your negotiating agent** to do negotiations for you. In the assignment you will use a software environment called GENIUS that can run agents that are written in Java. GENIUS is a negotiation environment that implements an open architecture for heterogeneous negotiating agents [13]. It provides a testbed for negotiating agents that includes a set of negotiation problems for benchmarking agents, a library of negotiation strategies, and analytical tools to evaluate an agent's performance.

**The assignment is split in 4 tasks** that will be described below. There will be two deadlines: First "D1" for task Task 1 and 2, and secondly "D2" for task 3 and 4. More details on the deliverables, requirements and deadlines are provided later in the document.

**Task 1: Create four preference profiles** Typically, negotiation is viewed as a process divided into several phases. Initially, in a pre-negotiation phase the negotiation domain and the issue structure related to the domain of negotiation are fixed. Negotiation may be about many things, ranging from quite personal issues such as deciding on a holiday destination to strictly business deals such as trading orange juice in international trade, or in this case a party. A negotiation scenario consists of a well structured negotiation domain (e.g. a set of negotiation issues and possible values for each issue) and preference profiles of the negotiation parties. We will provide you with such a domain, however your first task is to **create four realistic preference profiles** for this domain in GENIUS. This means translating your complex preferences to a few digital values that an agent will use to represent you. At deadline "D1" you should hand in these 4 profiles for the party domain, and a small report (pdf, max 1 page) describing the method you used to map a person's interests into a single utility function, and what the potential inaccuracies are. We advice to use the three group members as a source for the profiles.

**Task 2: build a (basic) negotiating agent** The second task involves thinking about a strategy used to perform the negotiation itself, i.e. the exchange of offers among your software agent and its opponents. It is worth noting that the negotiation we consider in this assignment is a closed multi-issue negotiation where there are multiple issues (i.e. more than one issue to be agreed on) and the negotiation parties only know their own preferences. It is not allowed to have access to other agents' preferences. The negotiation agent will follow the "**Stacked Alternating Offers Protocol**" as proposed in [1]. According to this protocol, all of the participants around the table get a turn per round; turns are taken clock-wise around the table, also known as a Round Robin schedule. One of the negotiating parties starts the negotiation with an offer that is observed by all others immediately. Whenever an offer is made, the next party in line can take the following actions:

- Accept the offer
- Make a counter offer (thus rejecting and overriding the previous offer)
- Walk away (thereby ending the negotiation without any agreement)

This process is repeated in a turn-taking clock-wise fashion until reaching a termination condition is met. The termination condition is met if a unanimous agreement or a deadline is reached, or if one of the negotiating parties ends the negotiation.

The following block will give some initial guidelines based on human experience with negotiation that may help you during your own negotiations and in building your own negotiating software agent.

- Orient yourself towards a win-win approach.
- Plan and have a concrete strategy.
- Know your reservation value, i.e. determine which bids you will never accept.
- Create options for mutual gain.
- Take the preferences of your opponents into account.
- Generate a variety of possibilities before deciding what to do.
- Pay a lot of attention to the flow of negotiation.

Some techniques relevant for building and designing negotiating agents can be found in section 2.4 among others in chapters 16 and 17 in [15]. The assignment will also require you to go beyond the course material in the book. Additional information is provided to you in the form of several papers [2, 10, 13, 6, 16]. There is a lot of other literature available about negotiation that may help you finish this assignment successfully. As for almost any subject, you can find more information about negotiation strategies online. We recommend to search for strategies used in the ANAC Competition [3, 4, 5, 8, 9,

11, 12, 17]. It is worth noting that some older ANAC agent strategies have been designed for bilateral negotiations where the agent has only one opponent, while in your assignment, your negotiating agent will negotiate with multiple opponent agents. Although you already know that your agent will negotiate with two agents, you are asked to **design and implement your agent in a generic way** so that it can negotiate with more than two agents too. In task 2 you must prove that you can build a functional basic agent by beating two naive 'random' agents that we provide. See the section 2.3 for more details on the requirements. We will ask you to **hand in this basic agent and the log of your result** by deadline "D1". *Remark: This basic agent is only a preliminary result because we expect an advanced and well motivated agent by deadline "D2"*

**Task 3: Analyze and improve your agent** The third task of the assignment takes place *during* the further development of this agent. We ask you to **provide us with your own detailed analysis of your agent's performance**. A good analysis method can help you to iteratively improve your agent, and shows us your understanding of factors that influence performance. The three parties that you **need to use** in your analysis are: `Atlas3 (ANAC2015-6-Atlas)`, `AgentBuyogMain (ANAC2015-13-Buyog)` and `Mercury (ANAC2015-5-Mercury)`. These are agents from the 2015 international agent competition that are provided within GENIUS. Of course you are allowed to use other agents *in addition* to these three. For deadline "D2", your report you must contain a section about the strong and weak points of your agent, together with graphs and metrics that support your analysis, a clear description of your testing (and improvement) method(s), and an explanation of why you choose these testing methods/metrics/graphs.

**Task 4: Hand in your final agent and report** Hand in your final agent so we can evaluate its negotiating performance against all other agents created by other groups. Part of your mark is dependent on this performance. Also you need to hand in a report explaining your final solution and your detailed analysis. Deadline "D2" is for handing in the report and agent.

The remainder of this document is organized as follows. In Section 2 the objectives, deliverables, requirements and assignment itself are described. Section 3 describes how to set up the GENIUS environment. Organizational details and important dates, including deadlines, are given in Section 4. Finally, Section 5 documents the evaluation criteria and grading for this assignment.

# 2 Detailed Assignment Description

The assignment must be completed in teams of 4 students. In the following paragraphs the objectives, deliverables, requirements, and the detailed assignment description are documented.

## 2.1 Objectives

- To learn to design and analyze a negotiating agent for a realistic domain with discrete issues, including among others a negotiation strategy.
- To learn techniques for implementing (adversarial) search and design heuristics while taking into account time constraints.
- To actively interact with other students and participate in student groups by discussing and coordinating the design and construction of a negotiating agent.

## 2.2 Deliverables

- [At deadline D1] Four preference profiles as .xml and a small report for task 1 (pdf, max 1 page) .
- [At deadline D1 and D2] A negotiating agent programmed in Java using the negotiation environment provided to you. Your submission should consist of a package containing all your agent source code

(java files, resources, etc). It is obligatory to use the package `ai2017.group`*n* where *n* is your group number. Please make sure all your files are in the directory structure as explained above. Assuming that the group has number three, a valid directory structure is:

```
ai2017/
    group3/
        Group3.java
        SomeHelperClass.java
```

- [At deadline D1, optional for D2] At D1 .log files of beating the naive agent in negotiation. You may also support your analysis in D2 with .log files of relevant results.
- [At deadline D2] A report in pdf format documenting and explaining firstly your solution and secondly your detailed performance analysis. The report needs not be lengthy (10 A4 pages may be enough, 15 A4 pages maximum), but should include an explanation and motivation of *all* of the choices made in the design of negotiating agent. The report should also help the reader to understand the organization of the source code (important details should be commented on in the source code itself). This means that the main Java methods used by your agent should be explained in the report. The report should also involve an elaborate analysis of your agent's performance from different perspectives (e.g. individual utility gained, social welfare - the sum of utilities of all agents, optimality of the outcome, fairness etc.).

## 2.3 Requirements

- The agent should make valid responses to the `chooseAction(...)` method.
- The agent should aim at the best negotiation outcome possible (i.e. the highest score for itself given the agent's preferences, while taking into account that it may need to concede to its opponents).
- The agent must take the utility of the opponents into account. Of course, initially you only have information about your own preferences. You will have to learn the preferences of the opponents during the negotiation process. The default way is by constructing an opponent model to have an idea of the utility of your proposed bids for the opponents.
- By deadline D1 your agent must be able to beat two random agents in a multi-party tournament. Beating is here defined as having the highest utility 15 out of 30 runs with the following tournament settings:

| | |
|---|---|
| Protocol | Stacked Alternating Offers Protocol |
| Deadline | 60 seconds |
| number of Tournaments | 5 |
| Agents per Session | 3 |
| Agent Repetition Allowed | no |
| Agents | `Random party`, `Random party2` and `your agent` |
| Profiles | `party1_utility.xml`, `party2_utility.xml`, `party3_utility.xml` |

- The report should include:
  - Your group number.
  - An introduction to the assignment.
  - A high-level description of the agent and its structure, including the main Java methods (mention these explicitly!) used in the negotiating agent that have been implemented in the source code.
  - An explanation of: the negotiation strategy, the decision function for accepting offers, and any other important preparatory steps and heuristics used for the bidding process.

- A section documenting the strong and weak points of your agent, the tests you performed to analyze (and improve) the negotiation strength of your agent. You must include scores of various tests over multiple sessions that you performed while testing your agent versus the three agents we provide. Describe how you set up the testing situation and how you used the results to modify your agent, and motivate why you choose these testing methods/metrics/graphs.
- A conclusion in which you summarize your experience as a team with regards to building the negotiating agent and discuss what extensions are required to use your agent in real-life negotiations to support (or even take over) negotiations performed by humans.

## 2.4 Designing an Agent

In this section, the main tasks and questions you need complete are presented. But before we do this, we present additional background that may help you to complete this assignment successfully. Note that the negotiation environment is a scientific software tool which is being improved constantly. Please report any major bugs found so that we can resolve them in a next version.

In this assignment a negotiation is also called a negotiation session. In a session agents negotiate with each other to settle a conflict and negotiate a deal. Each negotiation session is limited by a fixed amount of time. At the end of a session, a score is determined for both agents based on the utility of the deal for each agent if there is a deal, otherwise the score equals the reservation value; in this assignment 0.0. In a sense, there is no winner since each agent will obtain a score based on the outcome and its own utility function. A failed negotiation, in the sense that no deal is reached, thus is a missed opportunity for both agents. In the tournament that will be played, each agent will negotiate with all other agents and the scores of each session are recorded and averaged to obtain an overall score for the agent (see Section 5.1). A ranking will be compiled using these overall scores.

Key to this assignment is the fact that you have incomplete information about your opponents. You may assume that there is a conflict of interests, but at the start you do not know on which issues agents agree and on which they disagree. For example, in a negotiation about buying a laptop the buyer may prefer to have a middle-sized screen but the seller may prefer to sell laptops with small screens because (s)he has more of those in stock. They could, however, agree on the brand of laptop that they want to buy/sell. An outcome of a negotiation reconciles such differences and results in an agreed solution to resolve the conflict. Ideally, such an outcome has certain properties. One of these properties is that the outcome should be Pareto optimal. An outcome is Pareto optimal when there does not exist a deal where one agent can do better and one can do better or the same (i.e. they both score higher or equal, and therefore both would prefer this new deal over the old one). Another property is related to the Nash solution concept. A Nash solution is an outcome that satisfies certain bargaining axioms and may be viewed as a "fair outcome" which is reasonable to accept for both parties. An outcome is said to be a Nash solution whenever the product of the utility (in the range [0; 1]) of the outcome for the first agent and that for the second agent is maximal ([16]). In our case, the Nash product denotes the product of utilities for all agents. The notion of a fair outcome is important in negotiation because it provides a reference point for what your opponent might be willing to accept. Typically, a negotiation is started since reaching an agreement is better than not. But in order to get an agreement, you will need to get your opponents to agree. In a negotiation between self-interested agents, however, each agent is determined to get the best possible outcome for itself.

The Nash solution concept, as it is called, excludes certain outcomes as being unfair. It is, for example, very unlikely that your opponent will accept an offer that is most favorable to you and leaves your opponent with empty hands. In general, it is to be expected that an outcome is the result of a number of concessions that both parties make. Concessions can be made in various ways, quite easily or more slowly. The speed of making concessions, or the concession rate is the derivative of the (size of the) concession steps taken during a negotiation. An agent that makes concessions in very small steps initially uses a so-called Boulware strategy. A strategy that makes faster concessions initially is called a Conceder. Other strategies are conceivable and may be necessary given the deadlines (cf. [7])). To compute whether a bid in a negotiation is Pareto optimal or a Nash solution we use so-called utility functions (cf. [14]). In our case, utility functions assign quantitative values to

bids in the negotiation space. In this assignment, you may assume that all utility functions are additive, i.e. they will always be linear in the number of issues. For example, if there are four issues to negotiate about, the utility function can be computed by a weighted sum of the utilities associated with the chosen values for each of the issues. So, let $bid = \langle v_1, v_2, v_3, v_4 \rangle$ be a particular bid. Then the utility $u(bid) = u(v_1, v_2, v_3, v_4)$ (given weights $w_1, w_2, w_3, w_4$) can be calculated by: $u(v_1, v_2, v_3, v_4) = w_1 \cdot u(v_1) + w_2 \cdot u(v_2) + w_3 \cdot u(v_3) + w_4 \cdot u(v_4)$

The outcome space of a negotiation, i.e. all possible bids, can be plotted on a graph which indicates the utility of all agents on the y axes and the round numbers on the x axes. An example is provided in Figure 2.

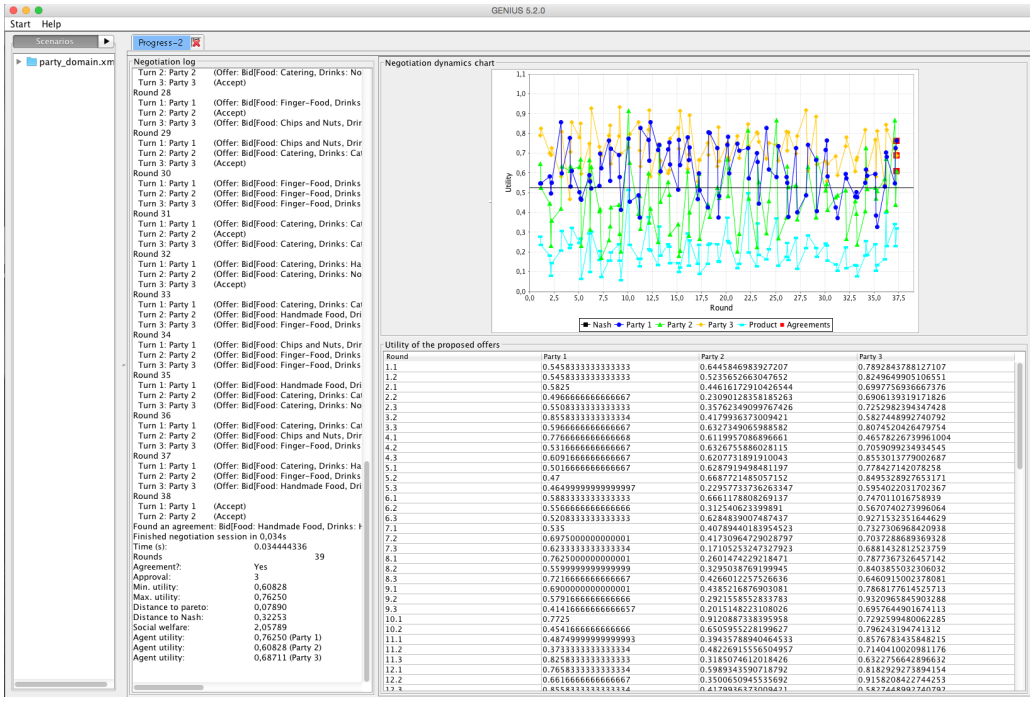


Figure 2: Trace of a three party negotiation

What sets the negotiations considered in this assignment apart from other negotiations is the fact that all agents have exactly the same deadline for achieving a deal and all agents "know" this.

## 3 Setting up the environment

Download the latest version of eclipse from

`https://eclipse.org/downloads/packages/release/Oxygen/R.`

You need the "Eclipse IDE for Java Developers", which will contain what we need for GENIUS development. To run GENIUS, download and unzip the latest version from

`http://ii.tudelft.nl/genius/?q=article/releases`

Then you can start GENIUS by running the `negosimulator.jar` file from the root folder.
A comprehensive guide for GENIUS can be found in the doc directory of the download, and also at:

`https://ii.tudelft.nl/genius/sites/default/files/userguide.pdf`

For this practical assignment we will focus on the options `Multi-Party Negotiation` and `Multi-Party Tournament`. The former sets up a single multilateral negotiation and displays it's trace, while the latter will run a tournament of multiple negotiations to get a statistically meaningful result.

The next step is to get GENIUS working in Eclipse so that you can start writing you own agent. Please note that, although GENIUS will work in most editors, Eclipse is the only editor officially supported for the practical assignment. This means that for other editors, we might not be able to help you if anything does not work.

You should also make sure that the assignment you handed in is runnable on our machines, which will run Eclipse and **Java 7**. Do not use Java 8 features.

A step-by-step guide on how to set up the project in Eclipse is given in the appendix at Section 6.1. New preference profiles can be created by right clicking the domain on the "Scenarios" tab in GENIUS. A step-by-step guide is given in the appendix at Section 6.2. To help you with the agent, we provide you with an example agent. Sections 6.3 and 6.4 in the appendix provide step-by-step guides on how to compile and rename the example agent and add it to GENIUS.

# 4 Organization

You may only use the e-mail address below to submit deliverables to the assignment or ask questions Important Dates:

1. Monday 18 September (pre-lecture): Deadline for registering your group. Please send an e-mail to **pa.ai-ewi@lists.tudelft.nl** listing your group members. We will assign a group number to you.
2. Monday 30 October, 23:59: Deadline D1. Submit a functional agent and preference profiles.
3. Friday 1 December, 23:59: Deadline D2. Submit the report and the final agent.

Please submit your report in PDF format and the package for your negotiating agent by mail to **pa.ai-ewi@lists.tudelft.nl**. Use the naming conventions described elsewhere in this document, including your group number. Do not submit incomplete assignment solutions; only a complete assignment solution containing all deliverables will be accepted. The deadline for submitting the assignment is strict. If you have problem with your agents, please contact us in advance.

# 5 Evaluation

Assignments are evaluated based on several criteria. Assignments need to be complete and satisfy the requirements stated in the detailed assignment description section above. Incomplete assignments are not evaluated. That is, if any of the deliverables is incomplete or missing (or fails to run), then the assignment is not evaluated.

The assignment will be evaluated using the following evaluation criteria:

1. Quality of the deliverables: Overall explanation and motivation of the design of your negotiating agent; Quality and completeness of answers and explanations provided to the questions posed in the assignment; Explanatory comments in source code, quality of documentation.
2. Performance: Agents will be ranked according to negotiating strength that is measured in a tournament (see also the next section below).
3. Originality: Any original features of the agent or solutions to questions posed are evaluated positively. Note that you are required to submit original source code designed and written by your own team. Source code that is very similar to that of known agents or other students will not be evaluated and be judged as insufficient. Detection of fraud will be reported to the administration.

## 5.1 Competition

Agents of teams will be ranked according to negotiation strength. The ranking will be decided by playing a tournament in which every agent plays negotiation sessions against a set of agents – including the agents programmed by your peers – with different sets of preference profiles. The testing will be done in a domain with discrete issues. An example of such a domain is the party domain.

Agents may be disqualified if they violate the spirit of fair play. In particular, the following behaviors are strictly prohibited: designing an agent in such a way that it benefits some specific other agent, starting new Threads, or hacking the API in any way.

### 5.2 Grading

Grades will be determined as a weighted average of several components. The grade for the practical assignment will be determined by your team solution (i.e. your assignment, the performance of your negotiating agent and report). The components that are graded and their relative weights are described in Table 1 below.

| **Assignment** | **Grading Method** | **Weight** |
|---|---|---|
| Negotiating Agent | Performance in a tournament with other agents. | 20% |
| Report | Agent design and the final report about your negotiating agent. | 80% |

Table 1: Grading criteria and weight

### 5.3 Master Thesis about negotiation

Did you like thinking about efficient negotiating strategies, or implementing a successful negotiating agent? Negotiation provides a lot of subjects to do your Master Thesis on! You can always ask or email the course assistants for more information: ai-ewi@lists.tudelft.nl

## Acknowledgements

This assignment could not have been written without the help of many people, including R. Aydogan, T. Baarslag, D. Festen, C. Rozemuller, L. van der Spaa, B. Grundeken, M. Hendrikx, K. Hindriks, C. Jonker, W. Pasman, D. Tykhonov, and W. Visser.

# 6 Appendix

## 6.1 Get Genius up and running in Eclipse

1. Open the Eclipse Navigator with the menu Window/Show View/Navigator. You can close the Package Explorer.
2. Right click in the Navigator area and select New/Java Project. Create a new Java project and name it `AI2017GroupNassignment`, where N is your group number. Make sure you select "JavaSE-1.7" or equivalent to ensure your code will be java 7 compatible **(Figure 3)**. Click Finish.

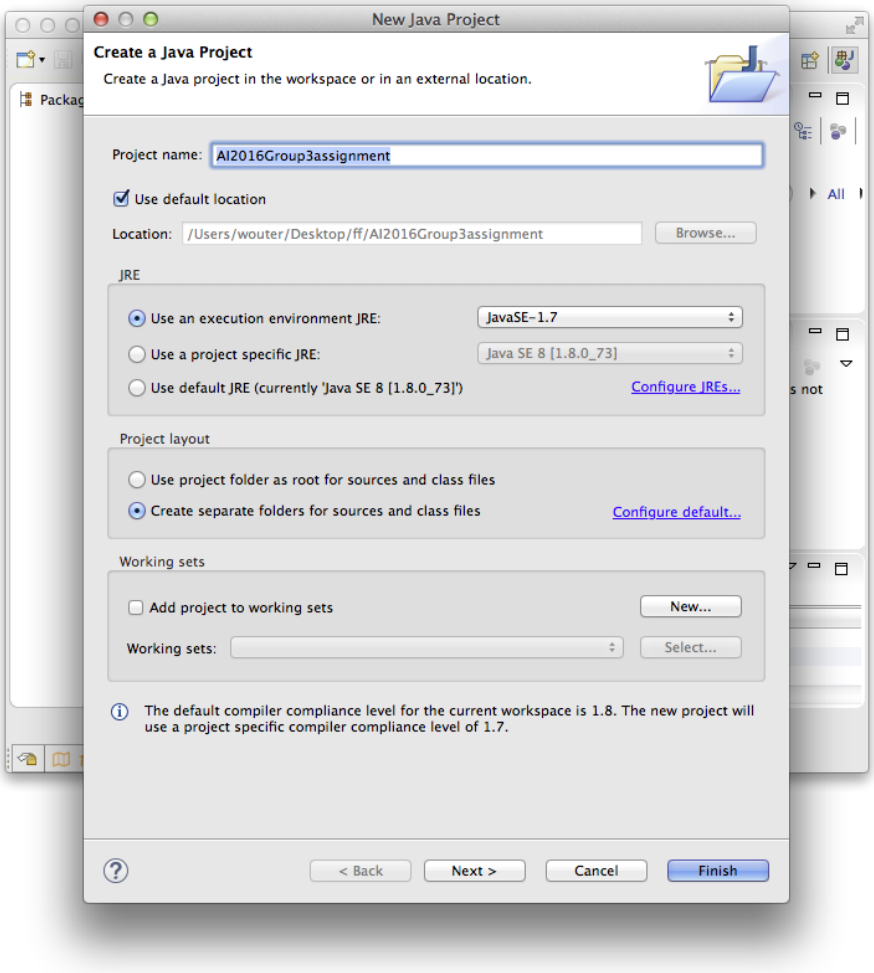


Figure 3: Create a new java project with the proper name and settings.

3. Drag all files from the genius folder into the project in the Eclipse Navigator area. Select "Copy files and folders" and press OK **(Figure 4)**.
4. Add the `negosimulator` Jar:
   (a) Right click on the `AI2017GroupNassignment` icon and select "Properties".
   (b) Select the Java Build Path.
   (c) Select the Libraries Tab.
   (d) Select "Add JARs", in the JAR Selection window **(Figure 5)**.
   (e) Open the `AI2016GroupNAssignment` folder and scroll down to select `negosimulator.jar`.
   (f) Click OK once.

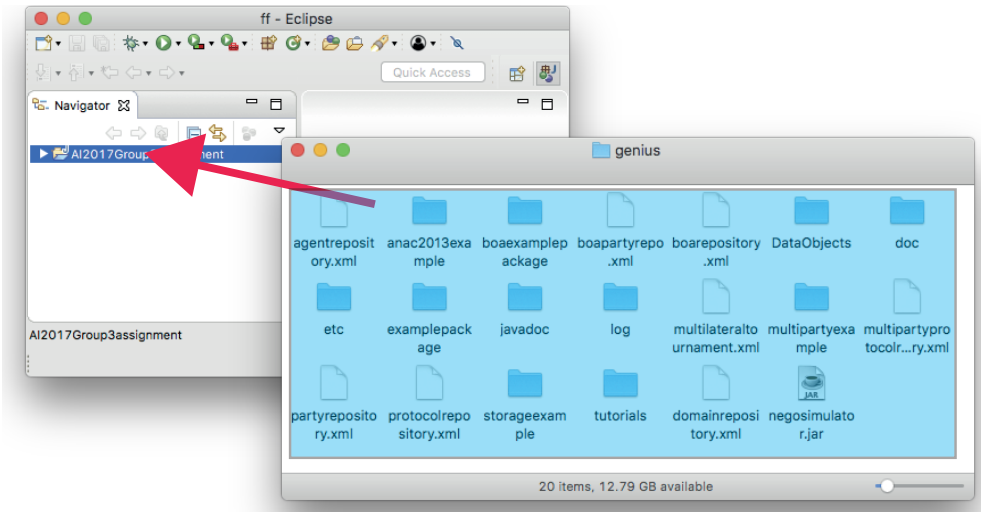


Figure 4: Copy Genius files

(g) Expand the library and double click "Javadoc location" **(Figure 6)**.

(h) click on the Browse button and select the "javadoc" directory inside your eclipse workspace.

(i) click a few times ok to close all dialog boxes.

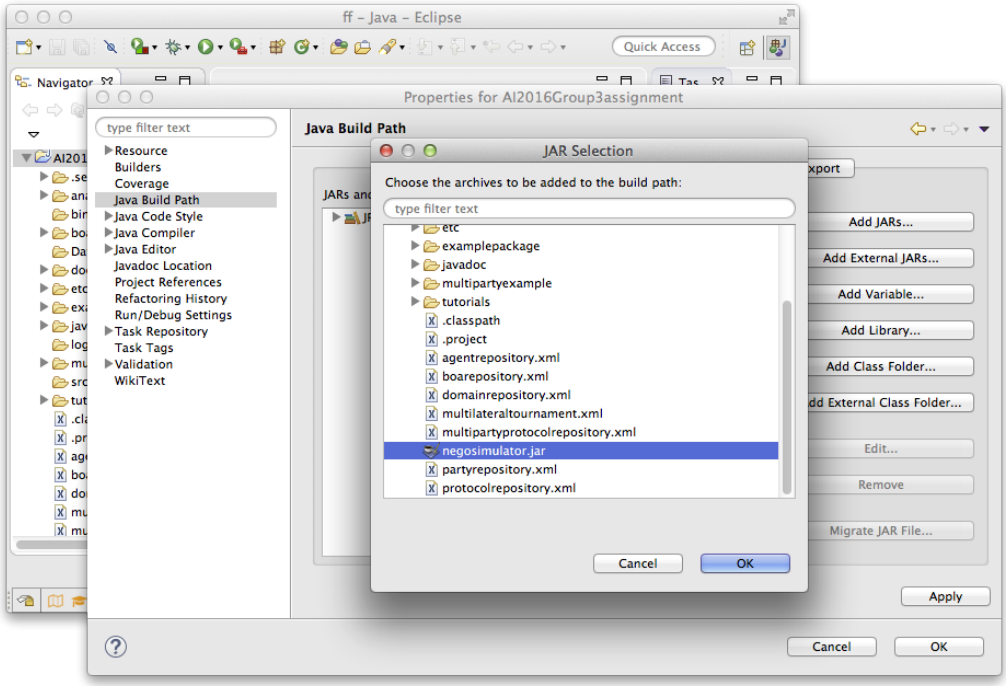


Figure 5: Attach the negosimulator jar to the project.

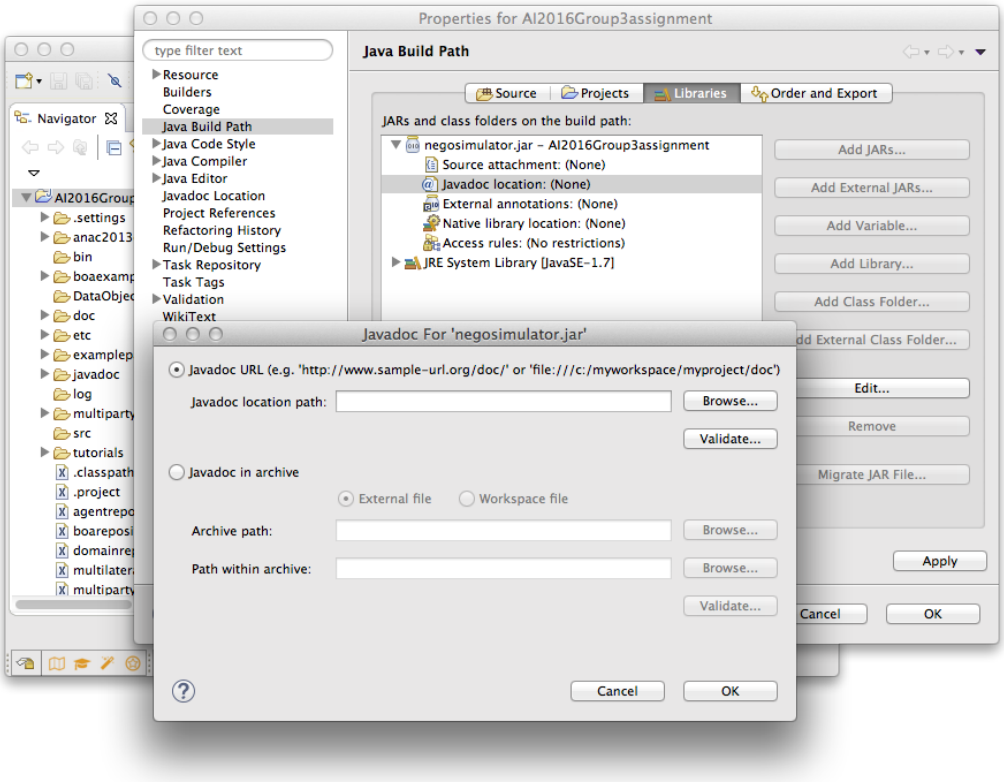


Figure 6: Attach the javadoc to the jar file.

5. Now you can run GENIUS as a Java application, by launching it as a `NegoGUIapp` **(Figure 7)**. To do this, right click on the project, select `Run As`, select `Java Application` and then in the browser select `NegoGuiApp`.

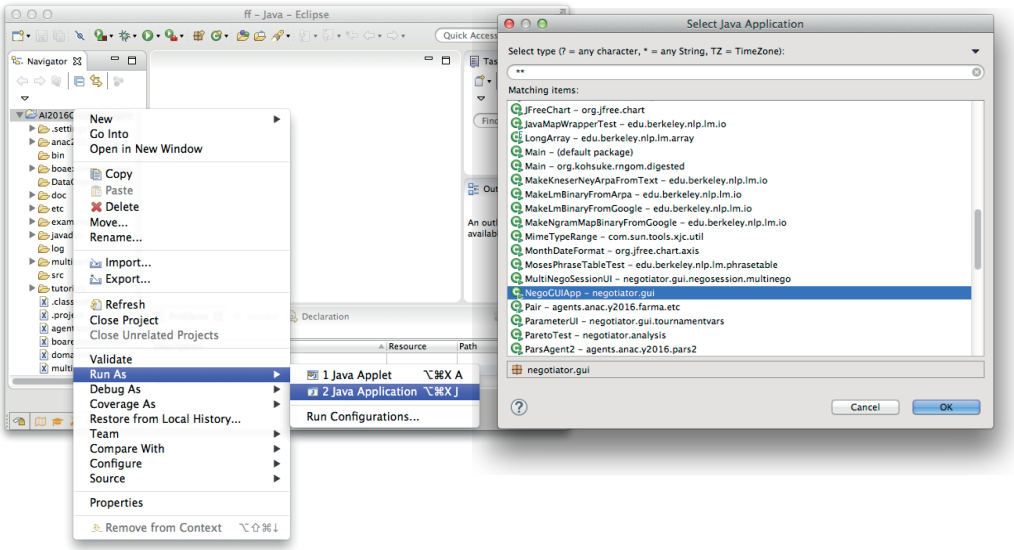


Figure 7: Starting Genius in Eclipse.

## 6.2 Create a negotiation profile in Genius

In GENIUS:

1. Right click the `party_domain` on the "Scenarios" tab.

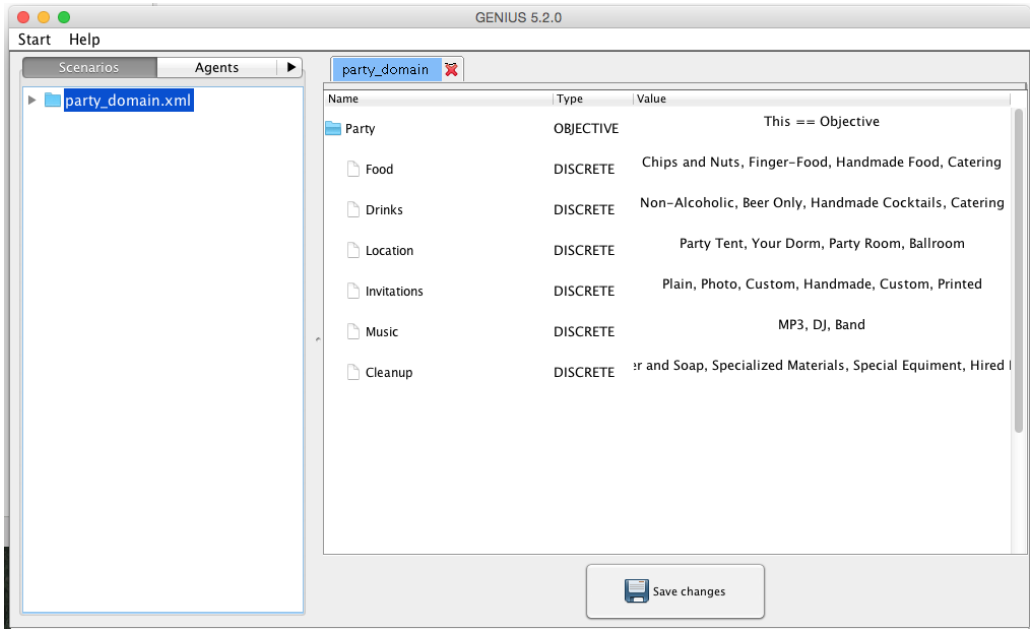


Figure 8: How to create a new preference profile - Step 1

2. Choose “New preference profile”.

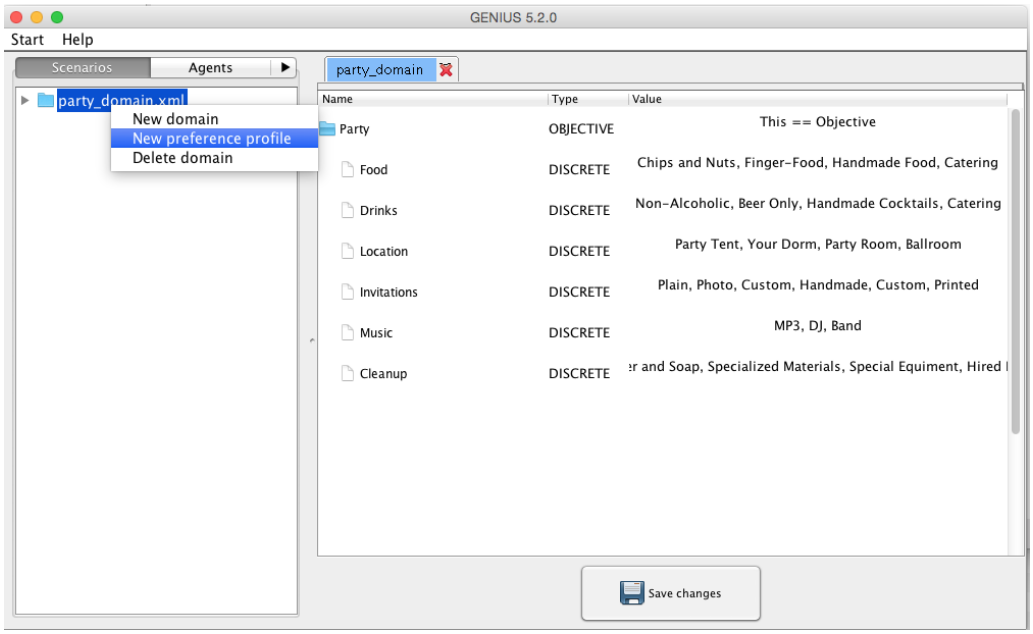


Figure 9: How to create a new preference profile - Step 2

3. Click each issue and set evaluation values for all possible options.

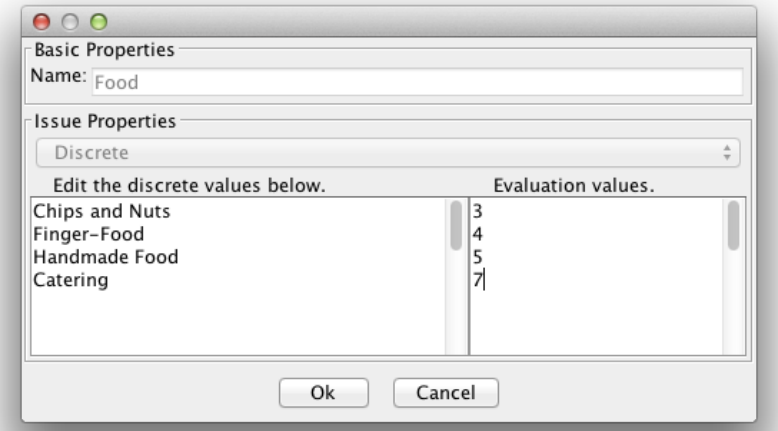


Figure 10: How to create a new preference profile - Step 3

4. Choose weights and click the "Save changes" button. Your profile is now saved in `etc/templates/partydomain`.

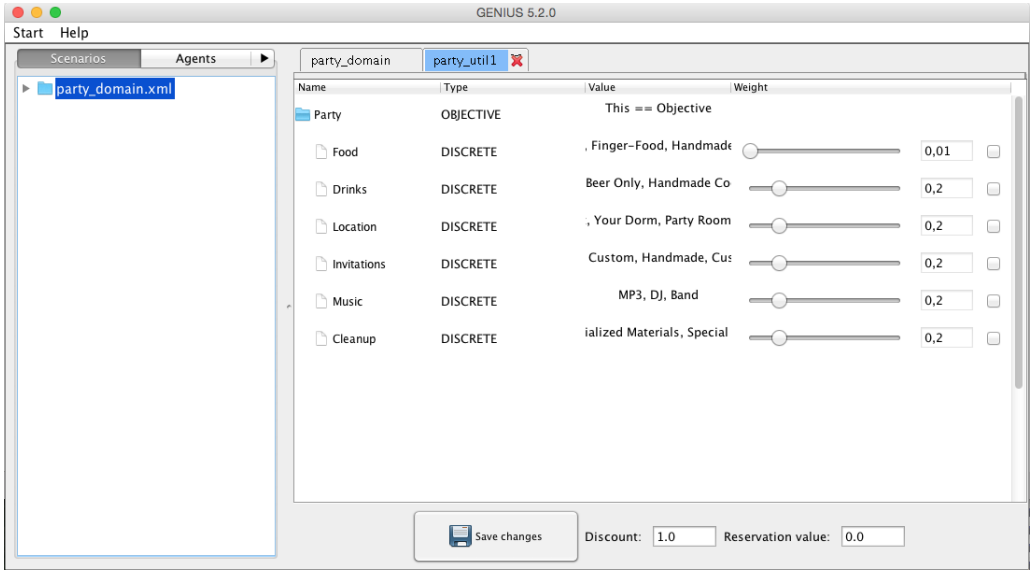


Figure 11: How to create a new preference profile - Step 4

## 6.3 Compiling the example agent for Genius

We provide one example agent for multiparty negotiation, however it is not compiled yet. Assuming you have set up GENIUS in Eclipse (appendix 6.1), this is how to compile and rename an agent in eclipse:

1. In the Genius Java project in Eclipse, copy the complete folder `multipartyexample` into the `src` directory.
2. Open Groupn.java inside src/multipartyexample.
3. Change the package name in the file to `ai2017`. Eclipse will now put a red line under it because it does not match the folder structure. Hover the mouse over it and select `Move 'Groupn.java' to package 'ai2017'.`

4. Change the class name `Groupn` in the file to match your actual group name. Eclipse will now put a red line under it because it does not match the file name. Hover the mouse over it and select `Rename compilation unit to 'Group3.java'`.

5. Save the file.

6. delete the now empty src/multipartyexample directory.

7. Your agent is now automatically compiled by Eclipse into the bin directory. The project should now look like Figure 12 in Eclipse.

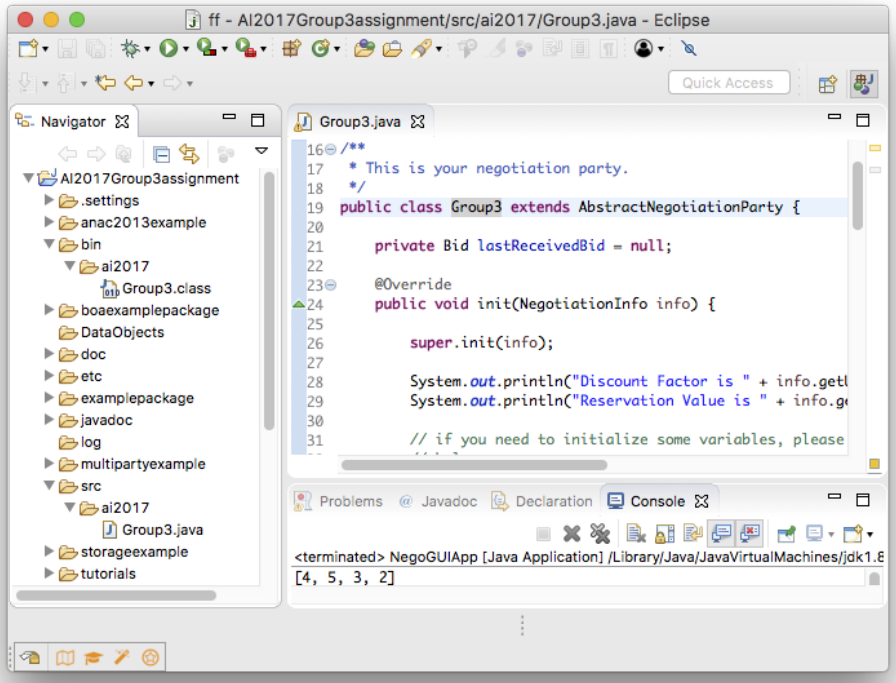


Figure 12: Set up the example agent to compile for your project

## 6.4 Adding an agent to Genius

In GENIUS:

1. Start up GENIUS

2. Select the Parties tab.

3. Right click in the Parties panel and select Add new party.

4. Browse to the compiled agent in your Eclipse project, in bin/ai2017/, and click OK (Figure 13).

5. Your agent should now have appeared at the end of the Parties list (Figure 14).

NOTE: if you change and re-compile your agent in Eclipse, these changes are not automatically picked up by Genius. This is because the way class loading in Java works. You have to quit and restart Genius to force a re-load of the agent.

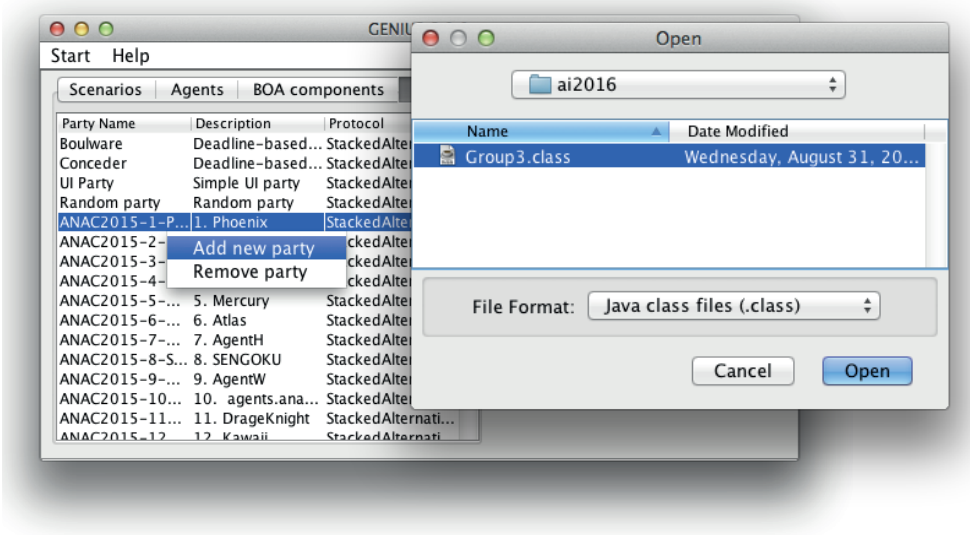


Figure 13: Your agent has been added to the list

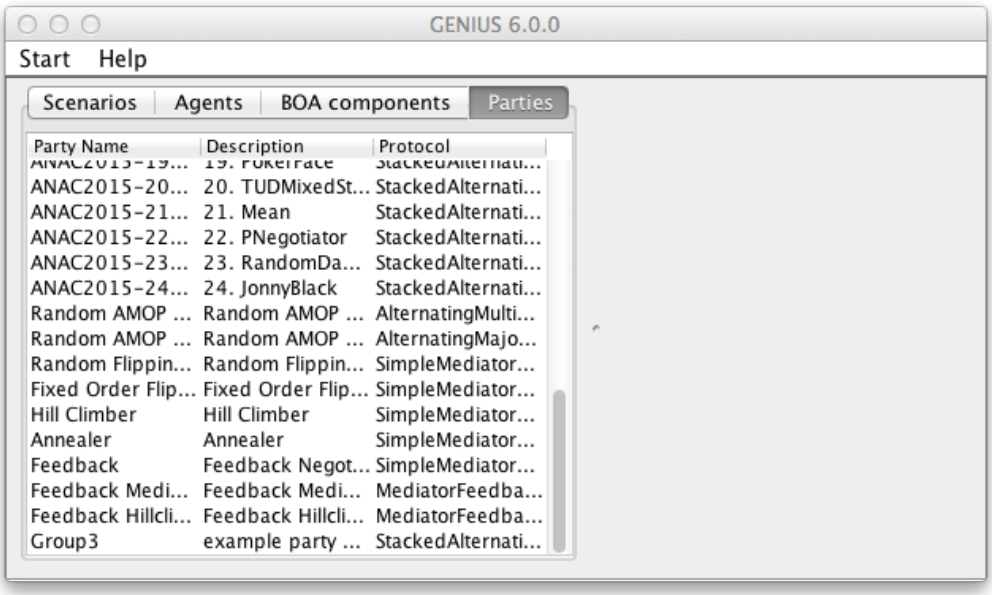


Figure 14: Your agent has been added to the list

# References

[1] Reyhan Aydoğan, David Festen, Koen V. Hindriks, and Catholijn M. Jonker. Alternating offers protocols for multilateral negotiation. In Katsuhide Fujita, Quan Bai, Takayuki Ito, Minjie Zhang, Rafik Hadfi, Fenghui Ren, and Reyhan Aydoğan, editors, *Modern Approaches to Agent-based Complex Automated Negotiation*. Springer. To be published.

[2] Tim Baarslag, Koen Hindriks, Mark Hendrikx, Alex Dirkzwager, and Catholijn Jonker. Decoupling negotiating agents to explore the space of negotiation strategies. In *Proceedings of The Fifth International Workshop on Agent-based Complex Automated Negotiations (ACAN 2012)*, 2012. URL = http://mmi.tudelft.nl/sites/default/files/boa.pdf.

[3] Tim Baarslag, Koen Hindriks, and Catholijn Jonker. A tit for tat negotiation strategy for real-time bilateral negotiations. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex*

*Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 229–233. Springer Berlin Heidelberg, 2013.

[4] Mai Ben Adar, Nadav Sofy, and Avshalom Elimelech. Gahboninho: Strategy for balancing pressure and compromise in automated negotiation. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 205–208. Springer Berlin Heidelberg, 2013.

[5] A.S.Y. Dirkzwager, M.J.C. Hendrikx, and J.R. Ruiter. The negotiator: A dynamic strategy for bilateral negotiations with time-based discounts. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 217–221. Springer Berlin Heidelberg, 2013.

[6] P. Faratin, C. Sierra, and N.R. Jennings. Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 24(3):159–182, 1998.

[7] S. Shaheen Fatima, Michael Wooldridge, and Nicholas R. Jennings. Optimal negotiation strategies for agents with incomplete information. In John-Jules Ch. Meyer and Milind Tambe, editors, *Proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*, pages 53–68, 2001. URL = citeseer.ist.psu.edu/fatima01optimal.html, November 2006.

[8] Radmila Fishel, Maya Bercovitch, and Ya'akov(Kobi) Gal. Bram agent. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 213–216. Springer Berlin Heidelberg, 2013.

[9] Asaf Frieder and Gal Miller. Value model agent: A novel preference profiler for negotiation with agents. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 199–203. Springer Berlin Heidelberg, 2013.

[10] C.M. Jonker and J. Treur. An agent architecture for multi-attribute negotiation. In *International joint conference on artificial intelligence*, volume 17, pages 1195–1201. LAWRENCE ERLBAUM ASSOCIATES LTD, 2001.

[11] Shogo Kawaguchi, Katsuhide Fujita, and Takayuki Ito. Agentk2: Compromising strategy based on estimated maximum utility for automated negotiating agents. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 235–241. Springer Berlin Heidelberg, 2013.

[12] Thijs Krimpen, Daphne Looije, and Siamak Hajizadeh. Hardheaded. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 223–227. Springer Berlin Heidelberg, 2013.

[13] Raz Lin, Sarit Kraus, Tim Baarslag, Dmytro Tykhonov, Koen Hindriks, and Catholijn M. Jonker. Genius: An integrated environment for supporting the design of generic automated negotiators. *Computational Intelligence*, 2012. URL = http://mmi.tudelft.nl/sites/default/files/genius.pdf.

[14] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

[15] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2010.

[16] Roberto Serrano. Bargaining, 2005. URL = http://levine.sscnet.ucla.edu/econ504/bargaining.pdf, November, 2005.

[17] Colin R. Williams, Valentin Robu, Enrico H. Gerding, and Nicholas R. Jennings. Iamhaggler2011: A gaussian process regression based negotiation agent. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 209–212. Springer Berlin Heidelberg, 2013.