Towards an Open Negotiation Architecture for Heterogeneous Agents

Koen V. Hindriks, Catholijn Jonker, and Dmytro Tykhonov

EEMCS, Delft University of Technology, Delft, The Netherlands
{k.v.hindriks, c.m.jonker, d.tykhonov}@tudelft.nl

Abstract. This paper presents the design of an open architecture for heterogeneous negotiating agents. Both the system level architecture as well as the architecture for negotiating agents are provided. The main contribution of this paper is that it derives a precisely specified interface from these architectures that facilitates an easy integration of heterogeneous agents into the overall negotiation framework. The interface is defined as a set of adapters that allows for various levels of integration of agents into the system architecture. The functionality provided by the system architecture depends on the number of adapters that are implemented and used to connect an agent to this architecture, ranging from functionality to conduct a bilateral negotiation to functionality for computing agent internal performance measures such as the quality of an opponent model. The architecture is used as the basis of a competitive testbed which allows us to study various negotiating agents. The design yields a flexible negotiation framework that facilitates negotiating different domains potentially using different protocols whereas no details of the internal negotiating agent structure are enforced. An application of the framework is illustrated by integrating two agents from the literature.

1 Introduction

The boost of literature on negotiating agents and strategies of recent years is in line with the continuous advance of ecommerce applications, such as eBay, and Marketplace in which negotiations play a role. While the literature focuses on the development of ever more clever negotiation agents [14, 6, 7, 17, 19, 11], the actual use of these agents in ecommerce applications is prohibited by two factors: the inflexibility of the agents and the lack of ecommerce applications that are open to such agents [13].

By the inflexibility of the agents we refer to the fact that they are incapable of negotiating with arbitrary agents and incapable of negotiating on arbitrary subjects. The code created for agents introducing new strategies in the literature typically has been developed with respect to one or a few specific domains, and to run against other agents implemented by the same team [14, 7, 17, 19]. This is understandable, since the code is to provide evidence of the excellence of the strategy. As a consequence, however, these agents cannot participate in a generic negotiation environment where heterogeneous agents can interact with each other. Interaction between such agents is not feasible due to several problems such as the absence of a shared negotiation ontology, and the lack of support for generic interaction protocols. Open negotiation environments and testbeds reported so far, such as the Trading Agent Competition [2], propose ontologies for a specific domain or scenario. The shared negotiation ontology must be generic to be able to model arbitrary negotiation domains.

Current as well as newly developed negotiating agents are (will be) written by different teams that should be free to select the technology of their choice to build such agents. In practice it is not possible to impose a particular coding and design standard for developing negotiating agents. The applicability of such agents, however, depends on their ability to interact in order to negotiate. Both the inflexibility of the current state-of-the-art negotiating agents and the closedness of existing ecommerce applications warrants the specification of a well-defined and precisely specified interface that allows such agents to conduct a negotiation.

Previous work on resolving these issues has focused mainly on the specification of generic interaction protocols [4, 23]. Our aim has been to design and implement a negotiation framework that allows existing heterogeneous agents to negotiate and to analyze the results of such negotiation. The framework should be able to function as a *testbed* as well as provide the *enabling technology for integrating heterogeneous agents*. To this end, an approach must be developed that enables the integration of arbitrary agents and algorithms for automated negotiation into a generic negotiation system architecture. In particular, an open system architecture for heterogeneous negotiating agents is needed, as well as a conceptually simple and generic agent architecture, in order to clarify the *requirements* on an interface to connect arbitrary negotiating agents to an overall system architecture that supports (bilateral) negotiation. Our choice to introduce an overall system architecture thus is motivated by several considerations: (i) it can be used to create a principled design of an interface enabling heterogeneous negotiating agents to engage in negotiation, (ii) it may be used as a testbed as well as for defining particular standards used to define a negotiation problem, and (iii) it precludes the need to specify ad hoc agent-to-agent interfaces. The architecture and interface developed in this paper provides the basis for an implementation of 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 and their strategies.

The paper is organized as follows. In Section 2 we propose an open architecture for heterogeneous negotiating agents and present a generic conceptual design of a negotiating agent architecture. Using this design an interface between agent and system architecture is specified. In Section 3 the adapters that are part of the interface are explained. The approach is illustrated by integrating two negotiating agents introduced in [17] and [19]. In Section 4 experiments are presented that demonstrate the usefulness of the environment as a testbed. Related work is discussed in Section 5. Section 6 concludes the paper.

2 Negotiation System and Agent Architecture

We introduce an architecture as a first step to a solution to the integration problem. The solution is applicable for integration of the existing agents as well as for the new agents that have not been implemented yet. This architecture has been implemented and provides the basis of our software negotiation framework. (This negotiation framework, user manuals, and a number of implemented negotiating agents can be downloaded from http://mmi.tudelft.nl/negotiation.)

Figure 1 illustrates the proposed architecture. The architecture is based on the analysis of the tasks of a generic negotiation environment. It represents a minimal but sufficient framework to enable integration of negotiation agents. The architecture consists of *four main layers* introduced below, a *human bidding interface*, and a *negotiating agent architecture*. An *interaction layer* is required to define and define the negotiation protocol and enable communication between agents. *An ontology layer* is needed to provide the actual functionality needed to define, specify and store a negotiation domain, the preferences of the negotiating agents. The architecture can be used for education and training of humans in negotiations. For that purpose, *a graphical user interface layer* provides options to create a negotiation ontology, defines agent preferences, allows human user(s) to participate in a negotiation, and review performance and benchmark results of agents that conducted a negotiation. *An analytical toolbox* is required to use the system as a research tool and organize tournaments. It provides a variety of tools to analyze the performance of agents and possibly internal quality measures related to e.g. the quality of an opponent model.

The overall architecture is introduced here to identify the main integration points where adapters are needed to connect a negotiating agent to this architecture. For the purpose of this paper, the human bidding interface is not relevant. The agent architecture itself identifies common components of a negotiating agent but is not intended to provide a comprehensive analysis of such architectures or go beyond the current state of the art [3, 8, 14]. This architecture may be instantiated with various software agents, which we illustrate below.

2.1 Negotiation System Architecture

Graphical User Interface The graphical user interface enables a user to define the negotiation game, i.e. the parameters of the negotiation, the subject or domain of negotiation, and preferences of agents (which also means that the preferences a human should take into account can be predefined). This interface does not introduce any integration points that should be part of the interface to integrate negotiating agents into the negotiation environment.

Negotiation Domain A negotiation domain is a specification of the objectives and issues to be resolved by means of negotiation. It specifies the structure and content of bids or offers exchanged, and of any final outcome or agreement (see also Fig. 4 and 5 below). An outcome determines a specific value for each issue, or, alternatively, only for a subset of the issues. Objectives allow to define a tree-like structure with either other objectives again or issues as children, in line with [22]. Various types of issues are allowed, including discrete enumerated value sets, integer-valued sets, real-valued sets, as well as a special type of issue called *price* issue. Additionally, a specification of a negotiation domain may introduce constraints on acceptable outcomes. For example, costs associated with a particular outcome may not exceed the available budget of the agent.

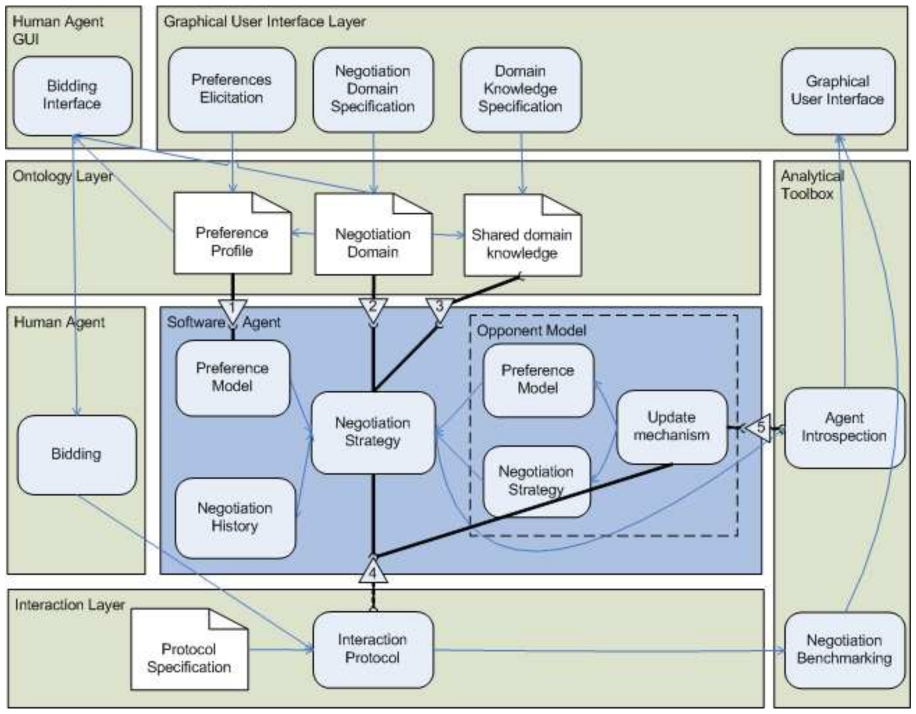


Fig. 1. The Open Negotiation System Architecture

Preference Profile A preference profile specifies the preferences regarding possible outcomes of an agent. It can be thought of as a function mapping outcomes of a negotiation domain onto the level of satisfaction an agent associates with that outcome. The structure of a preference profile for obvious reasons resembles that of a domain specification (see also Fig. 4 and 5 below). The tree-like structure allows to specify relative priorities of parts of the tree. This allows, for example, to ensure that all issues relating to travelling combined are weighted equally as all issues relating to the actual stay at a particular location.

Shared Domain Knowledge In a *closed* negotiation an agent is not informed about the preferences of its negotiating partner. In that case an agent can at best use a reconstruction (using e.g. machine learning techniques) of these preferences to decide on the negotiation move it should do next. It is typical, however, that with a domain comes certain public knowledge that is shared and can be used to obtain a better negotiation outcome. For example, common preferences such as preferring early delivery over later (though not always the case) may be common knowledge in a given domain. Such knowledge allows agents to compute the preferences of their negotiation partner e.g. using the time interval between two dates. This type of knowledge, labelled *shared domain knowledge*, is modelled explicitly as a separate component that can be accessed by all negotiating agents.

Interaction Protocol The interaction layer manages the *rules of encounter* or *protocol* that regulate the agent interaction in a negotiation [18]. Any agent that wants to participate in such a negotiation protocol must accept and agree to conform to these rules. An interaction protocol specifies which negotiation moves and what information exchange between agents is allowed during a negotiation. Interaction protocols are implemented in the negotiation environment as a separate component to allow the use of a variety of protocols [4]. The current version of the negotiation environment supports the alternating offer protocol [18], that allows a generic communication between the agents. The protocol is illustrated in Figure 2. A protocol can also dictate the exchange of complete package deal proposals or allow instead the exchange of partial bids. The layer also manages deadlines, or timeouts that may be set by the environment.

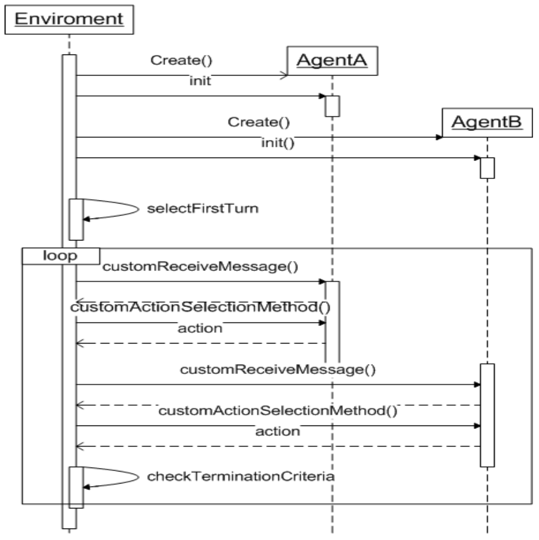


Fig. 2. A sequence diagram of the interaction protocol

The alternating offer protocol is not the only protocol used in the negotiation research. Therefore, the interaction protocols are implemented in the negotiation environment in a separate component to allow the use of a variety of protocols [4]. Implementation of a new interaction protocol in the negotiation environment is relatively easy task and has no or minimal effect on the agent code.

Analytical Toolbox The analytical toolbox layer of the architecture contains a set of statistical analysis methods to perform an outcome analysis on negotiation sessions as introduced and discussed in e.g., [10, 22]. Furthermore, the toolbox contains methods for the analysis of dynamic properties of negotiation sessions as discussed in e.g., [10]. The methods for both outcome and dynamics analysis were used to produce a number of performance benchmarks for negotiation behaviour and for the agent components [11]. The analytical toolbox uses the optimal solutions [21], such as the Pareto efficient frontier, Nash product and Kalai-Smorodinsky solution for the negotiation outcome benchmarking. The benchmarks in the negotiation system can be used to analyze the performance of opponent modelling techniques, the efficiency of negotiation strategies, and the negotiation behaviour of the agent. The result of the analysis can help researchers to improve their agents. The output of the analytical toolbox is presented by means of visualization (e.g., see 6).

2.2 Software Agent

The software agent component highlighted by the use of a different colour in Figure 1 is a generic component that can be instantiated by heterogeneous software agents. The components specified as part of a software agent in Figure 1 are part of the *conceptual design* of such agents but do not need to be actually present or identifiable as such in any particular software agent. These components do not introduce any design requirements for negotiating agents (although they could be used as such, see also [3]). Instead these components are introduced to identify *integration points* of agents with the system architecture. Five of such integration points, also referred to as *adapters*, have been identified.

The *preference model* component models the agent's preferences with respect to the negotiation outcomes. For example, the agents introduced in [11, 17, 19] use *utility functions* to represent preferences. Preferences however can be modeled by other structures, such as *ordinal rankings*. The *negotiation strategy* is the core component of a negotiating agent. This component makes decisions about the acceptance of an opponent's offer, ending a negotiation, and sending a counter-offer, using various tactics to generate such counter-offers [6]. The *negotiation history* component maintains the negotiation history, i.e. bids exchanged between agents, and can be used by the negotiation strategy component. It can also have a history records about earlier negotiations, the outcomes, identities of the opponents, and even opponent models. In repetitive negotiations with the same opponents this information can be used to improve negotiation performance of an agent by adapting the negotiation strategy and improving the opponent model.

In a typical negotiation setup preferences of the negotiating parties are private [22]. However, the efficiency of a negotiation strategy can be significantly improved by using information about opponent preferences [24]. Thus, an important component of a negotiating agent is an *opponent model*. Our generic component consists of three subcomponents: a preference model, a negotiation strategy, and an update mechanism. The component *preference model* contains representations of the preferences of the current and previous negotiating opponents. Typically, since the opponent's preferences are assumed to be private, the information stored in the component has a degree of uncertainty. The component *update mechanism* is used to interpret offers received from an opponent and to update the probability distribution associated with the preferences of an opponent. The purpose of the component *negotiation strategy* in the opponent model is to predict negotiation moves of the opponent. This knowledge can be used in the negotiation strategy to improve the efficiency of an agent's own offers and increase the chance of acceptance of an offer by the opponent. Models of the opponent's preferences and strategy are typically learned by the agent from negotiated agreements and offers exchanged in a negotiation [11, 24, 14].

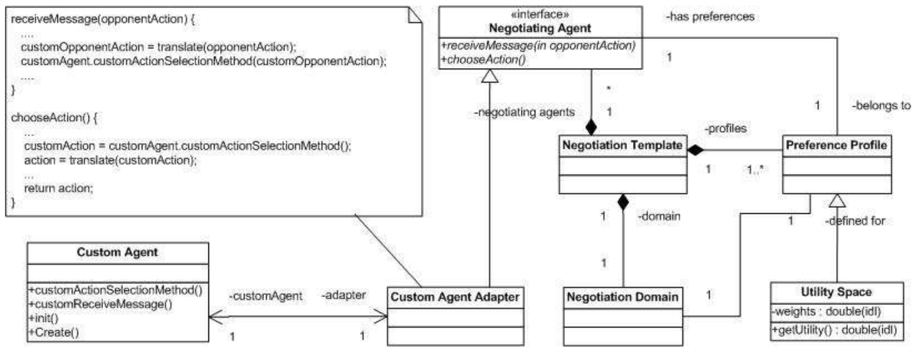
3 Interface and Adapters

To integrate heterogeneous negotiating agents in a single negotiation framework, their implementation has to be aligned with respect to the identified integration points of Figure 1. Alignment by redesign of an agent typically requires significant programming efforts and may cause back-compatibility problems. The number of adapters between agent to be developed in an *ad hoc* enviroment is quadratic in the number of agents: every pair of heterogeneous agents requires two adapters, one at each side. Wrapping agents and connecting them to a common framework requires only one adapter per agent, a number that is linear in the number of agents to be integrated. To minimize programming effort we propose a set of *adapters* or wrappers which need to be implemented once for each agent, and use software design patterns to develop these adapters [16]. From the five integration points identified, 3 must be implemented to be able to negotiate with another agent, including a negotiation domain adapter, a preference profile adapter and an interaction protocol adapter. Implementing the shared domain knowledge and agent introspection adapter provide additional functionality useful for more realistic negotiations, as well as for benchmarking agent performance.

In order to evaluate the effectiveness of our framework for integrating heterogeneous negotiating agents, we have integrated two existing agents from the literature, the *QO Agent* from [17] and the agent based on fuzzy modelling techniques [19] labelled *FBM* here. The integration of the agent should have no or minimal consequences for the performance of the agent. In order to validate that the integration did not affect the performance the integrated agent was evaluated and compared with the original implementation using the negotiation problems provided with that implementation. The results obtained did not show that the performance of the agents was significantly affected. Below we present the details and guidelines for implementing the adapters. Due to space limitations we cannot provide all details but only provide some specific findings regarding the integration of the *QO agent*.

Interaction Protocol Adapter The negotiation framework provides a skeleton Java class, called *Negotiating Agent*, to facilitate the implementation of a custom-made negotiating agent (see Figure 3). This class implements basic functionality of the agent such as the agent initialization, and the loading of a negotiation domain and preference profile, etc.

One of the most important tasks of this class is to ensure that a custom-made agent will comply with the negotiation protocol. The NegotiatingAgent class declares several methods that must be implemented in an agent. These



```
receiveMessage(opponentAction) {
  ....
  customOpponentAction = translate(opponentAction);
  customAgent.customActionSelectionMethod(customOpponentAction);
  ....
}

chooseAction() {
   ...
   customAction = customAgent.customActionSelectionMethod();
   action = translate(customAction);
   ...
   return action;
}
```

Fig. 3. A UML specification of the interaction protocol adapter

methods are called by the system architecture during a negotiation to inform an agent about its opponent's last action and to allow the agent to respond.

To integrate an existing agent in the negotiation framework we have used the Object Adapter design pattern. Figure 3 shows the adopted design pattern for the negotiating agent. In line with the pattern definition a Custom Agent Adapter class is added that is inherited from the Negotiating Agent class The receiveMessage() and chooseAction() methods of the adapter use the translation routines of the negotiation domain adapter.

Key to the successful integration of an existing agent is understanding the original code to a sufficient degree to understand the main information flows and interaction patterns. The main problem is the significant amount of time that is needed to analyze agent code to gain this insight. In particular, it is important to identify the agent's methods (a) that evaluate and interpret opponent bids and (b) that decide on the agent's next action. Moreover, differences in a protocol used by one agent from that of another require choices to be made as to what protocol to use in the negotiation framework. As an example, the protocol used by the original *QO Agent* is different from the alternating offers protocol and we chose to use the alternating offers protocol in our experiments.

Negotiation Domain and Shared Domain Knowledge Adapters This adapter must be able to interpret the information about the domain such as the number of issues, type of the issues and the values of the issues. Figure 4 shows a class diagram of the negotiation domain implementation of our negotiation framework. The Negotiation Domain class is a composition of a set of issues represented by the super class Issue. All classes for the specific issue types inherit from the Issue super class. Issues can be grouped into a hierarchical structure using the Objective class. As we explained earlier, our system provides four different types of issues: discrete, real, integer, and price issue (see the left part of the Figure 4 for the corresponding classes). Issues and corresponding values are bounded in the Bid class. An object of the Bid class represents one of the possible outcomes of a negotiation domain. The system implements a number of consistency checks to ensure that a bid is valid given the domain specification.

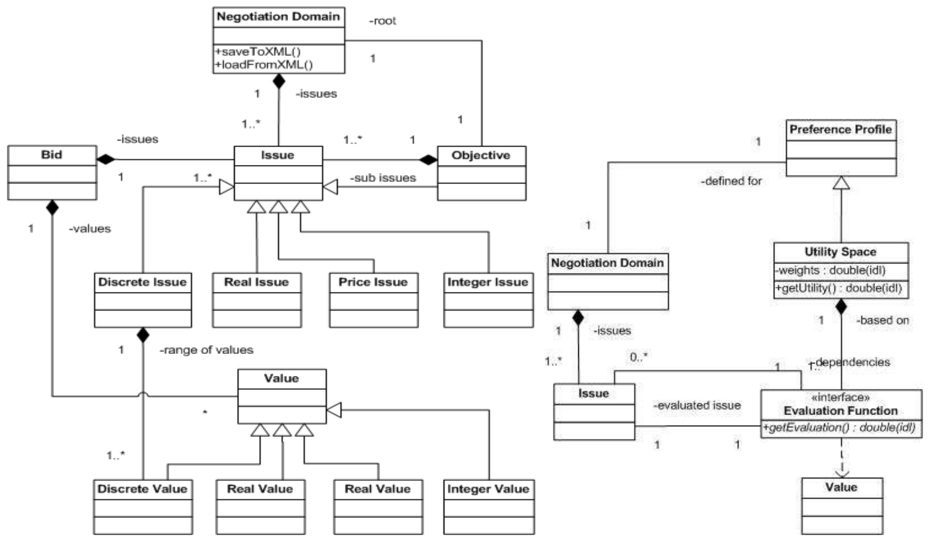
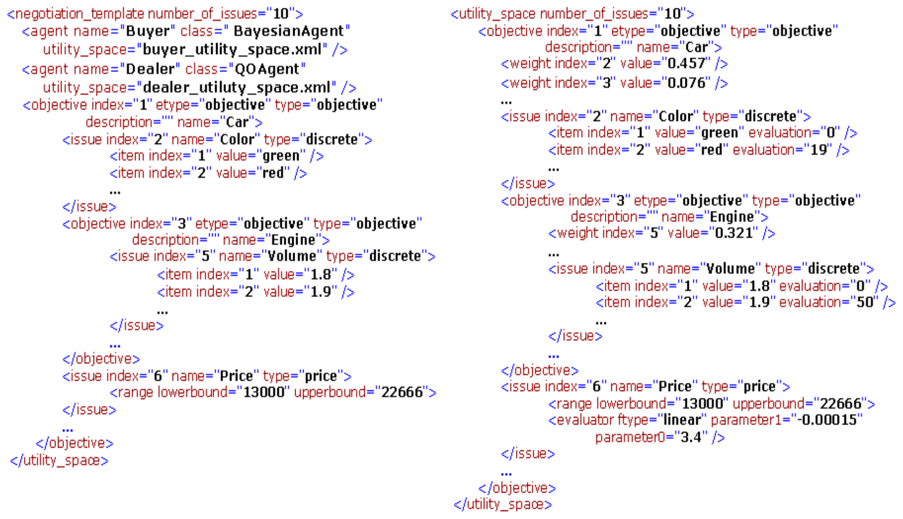


Fig. 4. Class diagram of the negotiation domain (left) and preference profile (right)

The adapter is implemented by two routines that translate the domain model provided by the negotiation system into the internal representation of the agent and vice versa. These routines are used to load a negotiation domain into the agent, interpret an incoming proposal from an opponent and generate negotiation moves.

A negotiation domain is represented by using a particular negotiation ontology. The negotiation ontology we use is specified in terms of XML files [1], which is widely-accepted file format for information exchange. Specific tags in the XML file correspond to each of the various Java classes used to store a negotiation domain. For example, the Objective class is represented with the tag labelled “objective” in the XML file that contains a negotiation domain specification (see the left part of the Figure 5). This tag can nest child tags such as other objectves and issues to represent the hierarchy as explained above. The *type* attribute of the *issue* tag specifies the type of the issue, such as discrete, real, integer. Discrete issues are defined by the *item* tags. An *item* tag defines a possible value of the issue in the *value* attribute. Intervals for the real and integer issues are defined using the lowerbound and upperbound of the *range* tag. The shared domain knowledge is also encoded in the negotiation domain XML file. The semantics of the domain knowledge must be interpreted by the agent itself, but is in fact supplied by means of the Negotiation Domain class which defines an XML document object model (DOM) [1].

Obviously, the routines to be developed mapping the ontology of the negotiation framework onto that of the negotiating agent and vice versa need to take the expressivity of the resepective ontologies into account. Only those parts of the ontologies can be mapped on each other that express the same meaning. Therefore the implementation of the negotiation domain adapter requires a careful analysis of the negotiation ontology of the original agent implementation first. Using this analysis the functions that translate negotiation concepts from the ontology used by the agent and the ontology used by the system architecture (see Fig. 5) have to be implemented. This procedure rather straightforwardly could be applied to the *QO agent*, which uses issues that can take discrete values, and uses plain text files to store a negotiation domain. Since discrete issues are one type of issue allowed by the negotiation system it is easy to define the required adapter methods.The method the *QO agent* needs to read a negotiation domain from an XML file (the format used by the system architecture) was wrapped up in a negotiation domain adapter.



```
<negotiation_template number_of_issues="10">
  <agent name="Buyer" class=" BayesianAgent"
     utility_space="buyer_utility_space.xml" />
  <agent name="Dealer" class="QOAgent"
     utility_space="dealer_utiluty_space.xml" />
  <objective index="1" etype="objective" type="objective"
          description="" name="Car">
      <issue index="2" name="Color" type="discrete">
            <item index="1" value="green" />
            <item index="2" value="red" />
            ...
      </issue>
      <objective index="3" etype="objective" type="objective"
              description="" name="Engine">
            <issue index="5" name="Volume" type="discrete">
                  <item index="1" value="1.8" />
                  <item index="2" value="1.9" />
                  ...
            </issue>
            ...
      </objective>
      <issue index="6" name="Price" type="price">
            <range lowerbound="13000" upperbound="22666">
      </issue>
      ...
    </objective>
</utility_space>
```

```
<utility_space number_of_issues="10">
    <objective index="1" etype="objective" type="objective"
              description="" name="Car">
       <weight index="2" value="0.457" />
       <weight index="3" value="0.076" />
       ...
       <issue index="2" name="Color" type="discrete">
              <item index="1" value="green" evaluation="0" />
              <item index="2" value="red" evaluation="19" />
              ...
       </issue>
       <objective index="3" etype="objective" type="objective"
                 description="" name="Engine">
              <weight index="5" value="0.321" />
              ...
              <issue index="5" name="Volume" type="discrete">
                     <item index="1" value="1.8" evaluation="0" />
                     <item index="2" value="1.9" evaluation="50" />
                     ...
              </issue>
              ...
       </objective>
       <issue index="6" name="Price" type="price">
              <range lowerbound="13000" upperbound="22666">
              <evaluator ftype="linear" parameter1="-0.00015"
                     parameter0="3.4" />
       </issue>
       ...
    </objective>
</utility_space>
```

Fig. 5. XML specification of a negotiation domain (left) and preference profile (right)

Preference Profile Adapter This adapter must be able to interpret the preferences of the agent as specified in Figure 4. The current implementation of the negotiation system operates with a preference structure based on utility functions. Other preferences modeling techniques, such as an ordinal ranking of the outcomes can be implemented by inheriting from the Preference Profile class. The utility space class in the negotiation system calculates the utility of an outcome as a weighted sum of the evaluations values of the individual issues, i.e. it implements linearly additive utility functions. The same type of utility functions are used by the *QO Agent*.

The preference profile, as the negotiation domain, can be saved as an XML file (see Figure 5). The structure of a preference profile XML file is similar to and extends the negotiation domain XML file with information about issue evaluators and their associated weights (priorities). The type of the evaluator is specified using the *type* attribute of the objective and issue tags. For example, for a discrete issue utility values are specified in the *evaluation* attribute of the *item* tag that represents the value of that issue. Consistency of a preference profile given a corresponding domain is checked automatically when it is loaded from the XML file by the negotiation system.

The procedure for implementing the preference model adapter is similar to that of implementing the negotiation domain adapter. As before, the representation of the agent's preferences in the original implementation need to be analyzed. In addition, one should verify whether the structure of the utility space and evaluation functions of the negotiation framework can be used to model the structure of the preferences in the original implementation. Since these aspects match for the *QO agent* the adapter could be implemented without much problems.

Agent introspection adapter The negotiation system architecture can be used as a testbed and research tool because it provides a number of benchmarks and tools to analyze negotiation performance [11]. To facilitate such analysis, an introspector is provided by the negotiation system. Negotiating agents can notify this introspector about a variety of events, such as an update of the opponent model, the selection of a next negotiation move, etc. The introspector must be allowed access to some of the internal structures of an agent such as its preference profile, its opponent model, and its negotiation history to be able to fully perform its function. This access is required to compute e.g. metric distances between an opponent preference model constructed by the agent and the actual preferences of the opponent.

We use the Observer pattern which is an event-driven design pattern to implement the introspector functionality. An agent needs to register with the introspector that plays the role of Observer. Components of the agent then need to notify the introspector when a corresponding event appears, which are subsequently logged and analyzed to obtain benchmarks.

Dedicated code must be written to be able to have the introspector compute some relevant performance measuresfor a particular agent. For example, for an agent that tries to learn the opponent's preferences measures related to the quality of learning as proposed in [12] can be computed. In order to do so, it is most important to locate those places in the agent code that can be used effectively for notifying the introspector to (re-)calculate the performance measures.

Lessons Learned The expressive power of the ontologies available for the specification of the negotiation domain and preference profiles were sufficient to express all possible options to define a domain and profile for both the *QO agent* as well as the *FBM agent*. The preference profiles of both agents are implemented as utility functions and the evaluation functions used by the agents to evaluate the values of the issues were also already present in our system architecture. The implementation of the adapter for the interaction layer, however, was more complicated than expected. The main reason is that the interaction protocol used by the original *QO agent* extends the alternating offers protocol since it allows additional types of messages to be exchanged between agents, such as threats. Moreover the localization of the core functions of the agent needed by the interaction protocol adapter determined most of the integration efforts. Ideally, therefore, an existing agent is integrated in close cooperation with its original developer. This is not an issue, however, if agent are implemented from scratch.

4 Experiments

One of the purposes of the proposed architecture is to allow for integration of heterogeneous agent and to facilitate comparison of their negotiation effectively as a testbed, and can be used to perform experiments with various negotiation domains, preference profiles and negotiating agents. The framework thus contributes to automated negotiating agents research by providing a tool that is able to provide new insights about such agents.

A tournament is a typical experimental setup for negotiating agents [10] that allows to measure success of an agent compared to the performance of the other. In addition, it is a useful tool to study the influence of various factors on the negotiation performance [12]. The analytical toolbox of the framework can generate a tournament setup given a set of agents, negotiation domains and preference profiles.

A small and simple negotiation problem, called "Party" [12], is used to analyze the performance of the *QO agent* within our negotiation framework. This domain has been created for negotiation experiments with humans, which also explains its rather limited size, including only 5 discrete issues with 5 possible values each (totaling to 3,125 possible outcomes). All of the issues are unpredictable [12], i.e. there is no shared domain knowledge. The preference profiles for the experiment were selected randomly from a set of 30 profiles created by human participants in a previously performed experiment. Since the *QO agent* needs 3 profiles as possible models of the opponent's preferences to be able to learn that profile, it was provided with the real profile of its opponent and two additional profiles that were randomly selected from all profiles.

In the experimental setup the *QO agent* negotiated against the Bayesian agent introduced in [11]. The Bayesian agent uses a learning algorithm using a Bayesian learning technique to build a model of the opponent's preferences. The techniques learns the necessary probabilities over a set of hypotheses about the evaluations and weights of the issues. Structural assumptions about the evaluation functions and weights are made to decrease the number of parameters to be learned and to simplify the learning task. Only one experiment was run for each combination of agents due to deterministic nature of the negotiation strategies of both agents.

Figure 6 presents the results of the negotiation experiment. The charts show the space of all possible negotiation outcomes. The axis represent the utilities of the outcomes with respect to the utility functions of the negotiating agents.

The charts show the negotiation paths of the agents marked by arrows with the names of the agents.

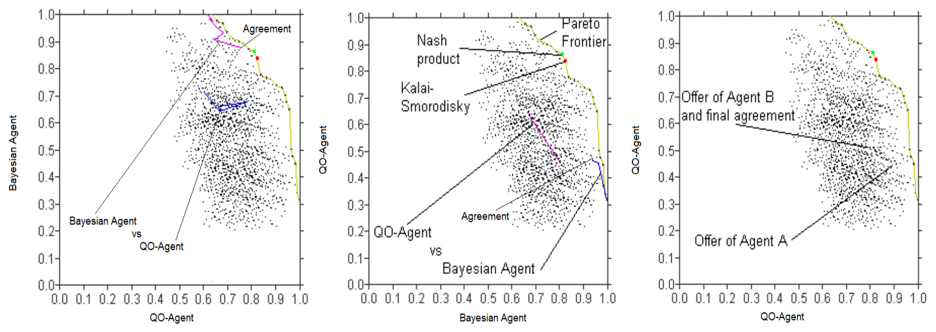


Fig. 6. Negotiation dynamics for the Party domain

The Bayesian agent starts with an offer that has maximum utility. It tries to learn the opponent preferences from the offers it receives and uses this model when it makes a concession towards the opponent. As a result, it stays close to the Pareto Efficient frontier. The *QO agent* in this domain has more difficulty to propose efficient offers. *This is a result of limitation of the opponent model of the agent.* The *QO agent* accepts an offer of the Bayesian agent as soon as such an offer has a utility level for the *QO agent* that is higher then utility of the *QO agent*'s own offer.

The other agent integrated into our negotiation system is the *FBM* agent introduced in [19]. The *FBM* agent was tested in a setup where it has to negotiate against the Bayesian agent about a single issue defined on real values ranging from 10 to 30. The original *FBM* agent is designed for negotiations where agents can exchange fuzzy proposals. The implementation of the *FBM* agent we used is able to negotiate about one-issue negotiations but can be extended for multi-issue negotiations. The agent adopts time dependent negotiation tactics from [6] and, thus, always makes concessions towards opponent. The offers are defined using two values: the peak value and the stretch of the offer. The preference profiles of the agents used were in complete opposition: the *FBM* agent wants to minize the value of the issues and the *Bayesian* agent tries of maximize it. In the experiments we performed, the β parameter that defines whether an agent makes bigger concessions in the beginning of the negotiation (Conceder) or at the end (Boulware) was varied, see Table 1.

In a single issue negotiation all negotiation outcomes are Pareto effient. The most important aspect of the negotiation strategy in a single issue negotiation is how fast one conceeds to the opponent. As a result, for $\beta > 1$ the *FBM* agent implements a Conceder tactic and the *FBM* agent undeperforms with respect to the *Bayesian* agent that makes linear concessions in this case because no moves towards the Pareto frontier are possible. When the *FBM* agent employs a

Table 1. Utility values of the FBM and Bayesian agents

Agents	Utility						
	β=0.02	β=0.1	β=0.5	β=1	β=2	β=10	β=50
FBM Agent	0.898	0.897	0.734	0.585	0.449	0.193	0.060
Bayesian Agent	0.102	0.103	0.266	0.415	0.551	0.807	0.940

Boulware tactic ($\beta < 1$) the *Bayesian* agent starts conceeding significantly and the result is a much lower utility for the *Bayesian* agent.

5 Related Work

There is a large body of related work available. Due to lack of space we cannot provide a complete overview but discuss specific approaches and examples of negotiation frameworks that allow us to clearly position our own work.

Generic frameworks for negotiation A range of quite different negotiation frameworks exist in the literature, including frameworks for (i) automated negotiating agents as well as (ii) negotiation support systems that provide electronic support for human negotiations. Within the first class a distinction can be made between argumentation-based systems, e.g. [8, 20], and heuristic-utility based systems, e.g. [10, 14, 17, 19]. The framework introduced and implemented belongs to the heuristic-utility based class of systems, though in principle it should be possible to use the framework for argumentation-based negotiation as well. Negotiation Support Systems (NSS) refer to systems that assist the process of human communication in negotiation, see, for example, [5, 15]. For example, in the Althena project (www.althenasoft.org) users can build content models, but the system does not support by means of predefined structures, repositories of content models, interaction support, or the selection of bidding strategies. Similarly, our framework, through a graphical user interface, allows users to create preference profiles, but significant extensions are needed in order to provide similar negotiation support useful for humans.

Architectures for negotiating agents The main focus in the literature on negotiating agent architectures, e.g. [3, 8, 14], is on the descriptive, structural and behavioural specification but not on the design of and requirements associated with interfaces. The system and agent architecture presented here are used specifically to obtain these interface requirements.

Negotiation ontologies Our work is related to work on negotiation ontologies to the extent that we need to define a language that can be used by heteregeneous agents to exchange offers. The language that we have used, based XML schemas, has been expressive enough to be able to integrate the *QO agent* and *FBM agent.* Our efforts to set up a negotiation ontology for the architecture proposed thus are motivated primarily by experience gained in practice. Related work about ontologies such as [4, 23] focus more on protocol than domain ontologies, and can be viewed as complementary. In future work our framework will be extended to handle negotiation ontologies for protocols as well.

Testbeds and Trading Competitions There is a variety of testbeds and trading competitions, but most are based on auction models instead of bilateral negotiation. In contrast, the framework introduced here provides a testbed that can be used to evaluate the behavior and performance of automated negotiating agents in bilateral negotiation. Moreover, most of the available testbeds are based on a specific domain, whereas we believe that there is a need for multi-issue bargaining testbeds which facilitate negotiation about various domains. A system somewhat similar to ours is the Neg-o-Net system [9]. Neg-o-Net is a generic agent-based computational simulation model for conducting multi-agent negotiations concerning resource and environmental management decisions, and includes a number of negotiation algorithms as well as agent models. However, the system is developed specifically to investigate environmental management.

6 Conclusion and Future Work

In this paper we have defined a clear interface for integrating bilateral negotiating agents into a (competitive) testbed. We have shown our approach is a viable and realistic one by demonstrating the actual integration of two negotiating software agents according to the "recipe" discussed in the paper. Some initial experimental results were provided to illustrate that by using the proposed environment we were able to easily obtain some new results and gain new insights on the current state of the art in the area of automated negotiation.

The negotiation environment described and developed based on the principles and specifications introduced here implements an open system architecture for such agents. The interface and adapters to connect agents to the negotiation environment have been clearly specified which enable an easy integration of heterogeneous negotiating agents. The actual use of this environment in, for example, ecommerce applications based on bilateral negotiation, however, is still significantly beyond its current main function as a testbed environment. Future research should pave the way to such applications, which would involve among others providing agents with the capability to propose a domain of negotiation, and to define the rules of the negotiation game (i.e., protocol selection). Additional research on ontologies for negotiation are required to make this feasible; for example, we cannot currently forumulate associated constraints on the domain of negotiation that must be satisfied for an agreement to be acceptable. More technically, components for web integration as well as extensions of adapters need to be developed, e.g., in order to handle more generic ontologies.

Acknowledgements

We would like to thank Raz Lin and Zaynab Raeesy for providing us with the code of their negotiating agents.

References

1. Extensible markup language (xml). http://www.w3.org/XML.

2. The trading agent competition. http://www.sics.se/tac.
3. R. Ashri, I. Rahwan, and M. Luck. Architectures for negotiating agents. In *The 3rd Int./Central And Eastern European Conf. on Multi-Agent Systems*, 2003.
4. C. Bartolini, C. Preist, and N. Jennings. A generic software framework for automated negotiation. Technical report, HP Labs, 2002.
5. E. Bellucci and J. Zeleznikow. A comparative study of negotiation support systems. In *Proceedings of HICSS*, 1998.
6. P. Faratin, C. Sierra, and N. R. Jennings. Negotiation decision functions for autonomous agents. *Int. Journal of Robotics and Autonomous Systems*, 24(3-4):159–182, 1998.
7. P. Faratin, C. Sierra, and N. R. Jennings. Using similarity criteria to make negotiation trade-offs. *Journal of Artificial Intelligence*, 142(2):205–237, 2003.
8. M. M. Geipel and G. Weiss. A generic framework for argumentation-based negotiation. In *Cooperative Information Agents XI*, volume 4676 of *Lecture Notes in Computer Science*, pages 209–223. Springer, 2007.
9. D. Hales. Neg-o-net - a negotiation simulation test-bed. Technical Report CPM-03-109, CPM, April 2002. Published as part of the FIRMA workpackage 3 report.
10. K. Hindriks, C. M. Jonker, and D. Tykhonov. Negotiation dynamics: Analysis, concession tactics, and outcomes. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 427–433, 2007.
11. K. Hindriks and D. Tykhonov. Opponent modelling in automated multi-issue negotiation using bayesian learning. In *Proceedings of the AAMAS 2008*, 2008.
12. K. Hindriks and D. Tykhonov. Towards a quality assessment method for learning preference profiles in negotiation. In *Proceedings of the AMEC 2008*, 2008.
13. N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1998.
14. C. M. Jonker, V. Robu, and J. Treur. An agent architecture for multi-attribute negotiation using incomplete preference information. *Journal of Autonomous Agents and Multi-Agent Systems*, 15(2):221–252, 2007.
15. G. E. Kersten and H. Lai. Negotiation support and e-negotiation systems: An overview. *Group Decision and Negotiation*, 16(6):553–586, 2007.
16. C. Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.* Prentice Hall PTR, 3 edition, 2004.
17. R. Lin, S. Kraus, J. Wilkenfeld, and J. Barry. Negotiating with bounded rational agents in environments with incomplete information using an automated agent. *Artificial Intelligence Journal*, 172(6-7):823–851, 2008.
18. M. J. Osborne and A. Rubinstein. *A Course in Game Theory.* MIT Press, 1994.
19. Z. Raeesy, J. Brzostwoski, and R. Kowalczyk. Towards a fuzzy-based model for human-like multi-agent negotiation. In *Proc. of the IEEE/WIC/ACM Int. Conf. on Intelligent Agent Technology*, pages 515–519, 2007.
20. I. Rahwan, S. D. Ramchurn, N. R. Jennings, P. McBurney, S. Parsons, and L. Sonenberg. Argumentation-based negotiation. *The Knowledge Engineering Review*, 18(4):343–375, 2004.
21. H. Raiffa. *The Art and Science of Negotiation.* Harvard University Press, 1982.
22. H. Raiffa, J. Richardson, and D. Metcalfe. *Negotiation Analysis: The Science and Art of Collaborative Decision Making.* Harvard University Press, 2003.
23. V. Tamma, S. Phelps, I. Dickinson, and M. Wooldridge. Ontologies for supporting negotiation in e-commerce. *Engineering Applications of Artificial Intelligence*, 18(2):223–236, 2005.
24. D. Zeng and K. Sycara. Bayesian learning in negotiation. *International Journal of Human Computer Systems*, 48:125–141, 1998.