Architectures for negotiating agents

Ronald Ashri[1], Iyad Rahwan[2], and Michael Luck[1]

[1] Dept of Electronics and Computer Science, Southampton University,
Southampton, UK
ra00r@ecs.soton.ac.uk
[2] Department of Information Systems, University of Melbourne
Melbourne, Australia
i.rahwan@pgrad.unimelb.edu.au

Abstract.

1 Introduction

In multi-agent environments, agents often need to interact in order to achieve their objectives or improve their performance. One type of interaction that is gaining increasing interest is *negotiation*. We adopt the following definition of negotiation that reconciles views proposed by [8] and [13], which we believe is a reasonable generalisation of both the explicit and implicit definitions in the literature.

> *Negotiation is a form of interaction in which a group of agents, with conflicting interests and a desire to cooperate, try to come to a mutually acceptable agreement on the division of scarce resources.*

Agents typically have conflicting interests when they have competing claims on scarce resources, which means their claims cannot be simultaneously satisfied. The use of the word "resources" here is to be taken in the general sense. Resources can be commodities, services, time, etc. which are needed to achieve something.

To address this problem, a number of interaction and decision mechanisms have been presented [3] and a number of implemented systems emerged. There has been extensive work on implementing frameworks of negotiation based on auction mechanisms as evident, for example, in the Trading Agent Competition (TAC) (cite). There is also a wealth of systems that adopt heuristic-based bilateral offer exchange (e.g. [6, 7]). Recently, argumentation-based approaches [9, 11, 1] have been gaining increasing interest. However, there are very few implemented systems that cater for this more sophisticated form of interaction. One of the reasons behind this is that many of these frameworks involve complex systems of reasoning based on logical theories of argumentation, for which there are yet many open research questions [12]. Another reason is that there are no software engineering methodologies that structure the process of designing and implementing such systems. This is why in most cases, these systems are implemented in an ad hoc fashion.

[3] For a more comprehensive comparison between different approaches to negotiation, the reader can refer to [8].

The aim of this paper is to address the software engineering issues related to the development of architectures for negotiating agents, ranging from simple "classical" agents to more complex "argumentative" negotiators. More specifically, this paper advances the state of the art in automated negotiation in the following ways. First, it presents a novel agent construction model that enables the description of a range of agent architectures through a common set of concepts. Secondly, it uses this agent construction model to describe the architectures of two generic classes of negotiating agents: *simple negotiators* and *argumentative negotiators*. More specifically, the paper demonstrates how a generic architecture for argumentative negotiators can be achieved by extending the simple negotiator architecture and reusing its components, and shows how this modularity is facilitated by the construction model.

The paper is organised as follows. We begin by presenting the agent construction model in section 2. In section 3, we present a generic architecture for a basic negotiating agent and explain how the construction model allow us to re-use it in developing a more complex architecture for an agent performing argumentation-based negotiation.

2 Engineering Agent Architectures

2.1 Design approach

In this section, we present the design approach that we use in specifying architectures for negotiating agents. The agent construction model should allow us to describe and compare a range of alternative architectures through a set of common concepts. In order to achieve this the construction model should be architecturally neutral. If the model already proposed a certain generic type of architecture (e.g. one inspired by belief-desire-intention) we would have to *translate* any non-generic architecture into a generic one. Such a translation, however, may lead to a loss of features that cannot be translated from one architecture into the other. Through a *truly* neutral model we can view a range of architectures based on a common understanding of agents and without losing in expressive capabilities. Furthermore, the model should allow for modular construction of agents. This is necessary both in order to meet general software engineering concerns but also the only way to delineate clearly the different aspects of an architecture, as we discuss below. Such a fine-grained approach leads to a better understanding of the overall functioning of the agent as well as how it can be altered. Finally, we need to be able to re-configure the resulting architectures easily, if possible even at run-time, in order to deal with dynamic, complex dependencies that develop in heterogeneous computing environments.

SMART The agent construction model departs from SMART [5] (Structured, Modular Agent Relationships and Types), which provides us with the foundational agent concepts that allow us to reason about different types of agents, and the relationships between them, through a single point of view. We chose SMART because it provides us with the appropriate agent concepts without restricting us to a specific agent architecture. Furthermore, SMART has already been successfully used to describe some existing agent architectures and systems [4, 3].

We avoid here a more complete presentation of SMART and focus on just those concepts that are used for the agent construction model. In essence, SMART provides a compositional approach to the description of agents that is based on two primitive concepts, *attributes* and *actions*. Formally, these primitives are specified as given sets which means that we say nothing about how they might be represented for any particular system. Attributes refer to describable features of the environment, while actions can change the environment by adding or removing attributes.

Now, an agent is described by a set of attributes and a set of *capabilities*, were capabilities are actions an agent can perform. An agent has *goals*, where goals are set of attributes that represent desirable states of the environment for the agent. On this basic concept of an agent SMART, adds the concept of an *autonomous agent* as an agent that generates its own goals through *motivations*, which drive the generation of goals. Motivations are can be preferences, desires, etc. of an autonomous agent that cause it to produce goals and execute actions in an attempt to achieve those goals.

This approach to agent description fits well with our requirement for architecture neutrality but does not sufficiently address our requirements for modularity and run-time reconfiguration so it is enhanced through a decoupling of the different aspects of an agent. This decoupling allows us to view agent from a *structural*, *behavioural* or *descriptive* point of view and as such improves the overall modularity of the resulting architecture and enables powerful run-time reconfiguration mechanisms. In the next section we elaborate on this refinements to the understanding of an agent.

Decoupling description, structure and behaviour In this subsection, we describe how we extend SMART to provide a more flexible decoupling of agent aspects. SMART allows systems to be specified based on an observer's point of view, based on their attributes and goals, as well as the actions they can perform. This description does not show how agents are built or how they behave. In other words, the focus is on the *what* and not the *why* or *how*. We call this a *descriptive specification*, since what it essentially does is describe the agent without analysing its underlying structures that sustain this description. Along with the descriptive specification we need to have the ability to specify systems based on their structure, i.e. the individual components that make up agents, as well as their behaviour. So we extend SMART with *structural specification* and *behavioural specification*.

The structural specification enables the identification of relevant building blocks and how different sets of building blocks enable the instantiation of different agent types. The behavioural specification of an agent addresses the process through which the agent arrives at such decisions as what actions to perform. These views, along with the descriptive specification, can provide a more complete picture of the system. The agent construction model, described next, reflects these concepts by allowing direct access to these different aspects of agents, based on a clear decoupling at the architectural level.

2.2 Agent construction model

The basic principles of the model are illustrated in Figure 1. A *shell* acts as the container in which *components* are placed. It manages the sequence in which components

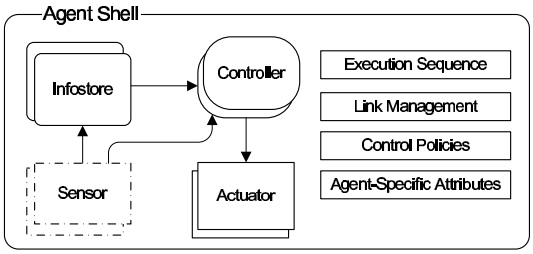

Fig. 1. Agent shell

execute and the flow of information between components. Control policies relating to the permissions an agent has in a specific environment are defined within the shell in order to make them independent of the agent architecture. Finally, attributes describing the agent as a whole are defined as part of the shell.

Components encapsulate specific types of actions that an agent can perform and are grouped into four categories. *Sensors* (illustrated as dotted line rectangles) receive information from the environment, *infostores* (rounded corner rectangle) store information, *actuators* (continuous line rectangle) perform actions that affect the environment and *controllers* (accented rounding) are the main decision-making components. Controllers analyse information and delegate actions to other components. The aim of dividing components into these categories is that it enables us to abstract between high level design, providing an understanding of an architecture early, before specific mechanisms for controllers, sensors or actuators have been defined. Each component is described using two types of attributes. *Stateless attributes* refer to persistent characteristics, such as the kind of communication language the agent uses, while *situation attributes* refer to attributes describing the component's current state (e.g. as the parties with whom the agent is currently negotiating with).

Information flows through *links* that the shell establishes between components. Links are uni-directional, one-to-one relationships. The information that flows through links between components is packaged within *statements*. One component acts as the producer of a statement and the other as the consumer. Statements are typed, and although currently just two types are defined, INFORM and EXECUTE, designers may choose to define different ones depending on the application needs. Inform-type statements are used when one component simply notifies another component about something; while execute-type statements are used when a component wants an action to be performed from another component. All statements are divided into a *body* and *predicates*. The body carries the main information (e.g. an update from a sensor), while the predicates carry additional information (e.g. the source of information or specific conditions associated with the execution of the action).

The sequence in which components execute is defined as the *execution sequence* of the architecture. Execution of a component includes the processing of statements received, the dispatch of statements and the performance of any other actions that are called for. The execution sequence is an essential aspect of most agent architectures

and, by placing the responsibility of managing the sequence within the shell, we can easily reconfigure it at any point during the agent's operation.

An agent design begins with an empty shell. It can then be specialised by defining control policies in order for it to meet application requirements or the demands of the environment within which it will operate. One could envisage implementations of shells being provided by environment owners, which would ensure compatibility with their environment while allowing the agent developer relative freedom in designing the structure and behaviour of the agent within that shell. Consequently, shell-specific attributes can be defined to form part of the description of the agent to the outside world. The components can then be loaded into the shell, and link relationships, as well as an execution sequence, can be defined. With the execution sequence in place, the operational cycle of the agent can begin. The agent lifecycle can be suspended or stopped by stopping the execution sequence. This operational cycle can be modified by altering the execution sequence, modifying relationships between components, or by applying alternative control policies.

One of the main benefits of this approach, is that it is possible at any moment to extract the three aspects of agent as described previously. The descriptive specification can be obtained by aggregating the situation attributes and stateless attributes from each component as well as the attributes contained at the shell level. The structural specification is given by the components and the behavioural specification is given by the execution sequence and the links between components.

3 Negotiating Agent Architectures

With the agent construction model in place we can now investigate the suitability of our model for specifying flexible negotiating agent architectures. But before we start describing negotiating agents, we discuss the main components of a *negotiation framework*. In addition to the negotiating agents, a negotiation framework usually includes a communication language and an interaction protocol. For example, a negotiation framework based on a simple English Auction protocol would need a communication language locution (or performative), say $propose(.)$, that can express bids. The protocol is the set of rules that specify, at each stage of the interaction, what locutions can be made, and by who. In addition, the framework needs a language for representing information about the world, such as agents, agreements, arguments, and so on. This information is used within the communication language locutions to form utterances. For example, a bid might be presented as $propose(a, b, \{toyota, \$10K\})$, where a and b are the sending and receiving agents , and $\{toyota, \$10K\}$ is the specification of the proposal. Finally, a negotiation framework usually includes various information stores needed to keep track of various information during the interaction. This information may include proposals made by different agents, concessions they have committed to [13], and so on. Finally, the framework also needs a set of additional non-protocol rules. These may include rules that identify the winner in a particular negotiation, or rules that specify that agents cannot retract their previous proposals, and so on.

In this paper, we focus our attention on the construction of the agents within the framework. So we do not address, for example, how protocols can be modularly speci-

fied (this has been investigated in [2] for example), or how the locutions can be verified. We assume that developers have at their disposal definitions of the appropriate negotiating protocols, domain ontologies and communication languages, and instead deal with the problem of framing such mechanisms within an appropriate agent architecture. Note that we do not claim to have specified the *only* way of describing negotiating agents. Instead, we attempted to provide *a* construction model that is generic enough to capture a variety of negotiators.

3.1 Basic Negotiating Agent

We begin by presenting a generic model of a *basic negotiating agent*, illustrated in Figure 2. Basic negotiating agents include those participating in auctions or those engaged in bilateral offer exchanges. The common aspect of these agents is that they engage in interactions in which the primary type of information exchanged between agents are proposals (i.e., potential agreements). We call them *basic* in order to distinguish them from agents that can engage in more sophisticated forms of negotiations which allows the exchange of meta-information (or arguments). We discuss the latter form of negotiators in the next subsection.

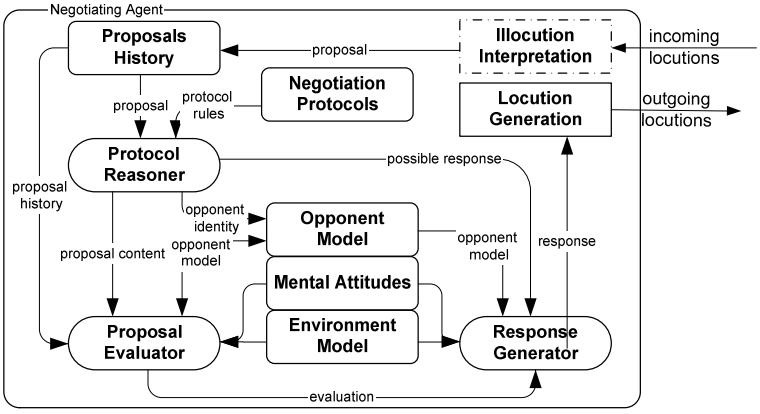

Fig. 2. Negotiating Agent Architecture

Descriptive Specification The description of the negotiating agent is based on its attributes, capabilities, goals and motivations. The goals of the agent, i.e. the desired negotiation outcomes, could be represented in the *Mental Attitudes* infostore, and would refer to specific application domains. The architecture, however, does not require explicit representation of agent goals. We could have as an overarching goal the achievement of the environmental state which represents the best possible negotiation outcome for the agent. This optimal state would be determined by the mechanisms used by the *Proposal Evaluator* and *Response Generator* components, which will ultimately decide

when this environmental state has been reached. Here we see how access to an overarching, architecturally neutral agent theory allows to reason about such things as goals even though they find no explicit representation in the architecture.

Attributes of the agent are given by the types of information that is stored and interpreted inside components and flows between them through statements. These attributes will include representations of beliefs about the opponents, the environment, mental attitudes, negotiation protocols, and so on. Because all this information is explicitly represented within components and stateless and state-dependent information is separated we can easily extract it.

The capabilities of the agent are given by the aggregation of capabilities of each component and can be understood, in our case, by referring to individual components in the architecture diagram. This is possible because our architecture attempts to represent the main capabilities with separate components so as to make clear the various functionalities required. However, alternative designs could (as in many implementations in the literature) combine a number of components (e.g. the representation of opponents, mental attitudes and environment) within a single component. In such cases, the descriptive specification would remain unaltered, since the capabilities exist, but the structural specification would refer to different components that combine those capabilities.

Finally, the motivations of the agent, if the agent were autonomous, form part of the agents mental attitudes, and ultimately guide the agent's decisions. How exactly these are defined depends on the application. In many auction-based mechanisms, for example, the motivations are represented in the form of a utility function.

Structural specification The structure of the agent refers to the components that make up the architecture. Messages are received, checked and parsed through the *Illocution Interpreter*. The *Proposals History* infostore keeps track of the various proposals received. The *Negotiation Protocols* infostore contains the rules relating to the negotiation protocols. By separating the rules dictating the protocol, from the reasoning about the protocol we can more easily extend the agent to handle different protocols. The *Opponent Model* infostore keeps track of opponents models, while the *Environment Model* keeps information about the environment within which the agent is situated. Information such as the agent's preferences in stored in the *Mental Attitudes* infostore. Decisions are taken by three controllers that, abstractly, support the different negotiation stages. The *Protocol Reasoner* checks whether the proposal received by the opponent is a valid response based on the negotiation protocol. The *Proposal Evaluator* evaluates the proposal and the *Response Generator* generates an appropriate response based on this evaluation. Finally, the *Locution Generator* packages responses in the appropriate message format and handles outgoing communication.

Behavioural Specification The behaviour of the agent is largely dictated by the flow of information through the architecture. It begins by message interpretation and storage in *Proposals History*. The current proposal and information of the history of proposals is send to the *Protocol Reasoner*, which uses rules in the *Negotiation Protocols* infostore to check the validity of the proposal. If it is valid it is forwarded to the *Proposal Evaluator*, which retrieves information about the opponent from the *Opponent Model* infostore. This controller uses this information along with information from the

Mental Attitudes, *Environment Model* and *Proposal History* to evaluate the proposal. As a result of the evaluation, the evaluation is send to the *Response Generator*, and the opponent model may be updated. This controller also uses information from the now updated opponent model, the mental attitudes and environment model in order to generate a response. It also takes into account the negotiation protocol rules in order to generate the appropriate response. The response is packaged in the appropriate format by the *Locution Generator* before it is sent to the opponent.

3.2 Argumentative Negotiating Agent

In this section, we reuse the architecture of the basic negotiating agent in order to provide a generic description of agents capable of conducting argumentation-based negotiation (ABN). An argumentative negotiator shares many components with the basic negotiator. For example, it also needs to be able to evaluate proposals, generate proposals and so on. What makes argumentative agents different is that they can exchange meta-information (or arguments) in addition to the simple proposal, acceptance, and rejection utterances. These arguments can potentially allow agents to (i) justify its negotiation stance; or (ii) influence the counterparty's negotiation stance [9]. This can potentially lead to (i) better chance of reaching agreement; and/or (ii) higher-quality agreements. In ABN, influencing the counterparty's negotiation stance takes place as a result of providing it with new information, which may influence its mental attitudes (e.g., its beliefs, desires, intentions, goals, preferences, and so on). This can potentially entice (or force) the agent to accept a particular proposal, or concede on a difficult issue. Arguments can range from threats and promises (e.g. cite Sierra) to logical discussion of the agent's beliefs (e.g. [11]) or underlying interests (cite Rahwan).

In order to facilitate ABN, the logical and communication language usually needs to be capable of expressing a wider range of concepts. For example, the proposal might instead by represented as $propose(a, b, P, A)$ where a and b are agents, P is a proposal, and A is a supporting argument denoting why the recipient should accept that proposal. ABN frameworks may also allow agents to explicitly request information from one another. This may be done, for example, by posing direct questions about agent's preferences or beliefs, or be challenging certain assumptions the agent adopts. Since in this paper we are more interested in the abstract structures within the agents, we shall not discuss these issues in more detail. In order to be capable of engaging in ABN, an agent needs the following additional capabilities:

1. **Argument Evaluation:** This component encompasses the ability of the agent to assess an argument presented by another, which may cause updates to its mental state. This is the fundamental component that allows negotiators' positions to change.
2. **Argument Generation:** This component allows the agent to generate possible arguments, either to support a proposal, or as an individual piece of meta-information. The locution generated may also be a question to present to the opponent.
3. **Argument Selection:** Sometimes, there might be a number of possible arguments to present. For example, an agent might be able to either make a promise or a threat to its opponent. A separate component is needed to allow the agent to choose the more preferred argument. Selection might be based on some analysis of the expected influence of the argument, or on the commitments it ties the utterer to.

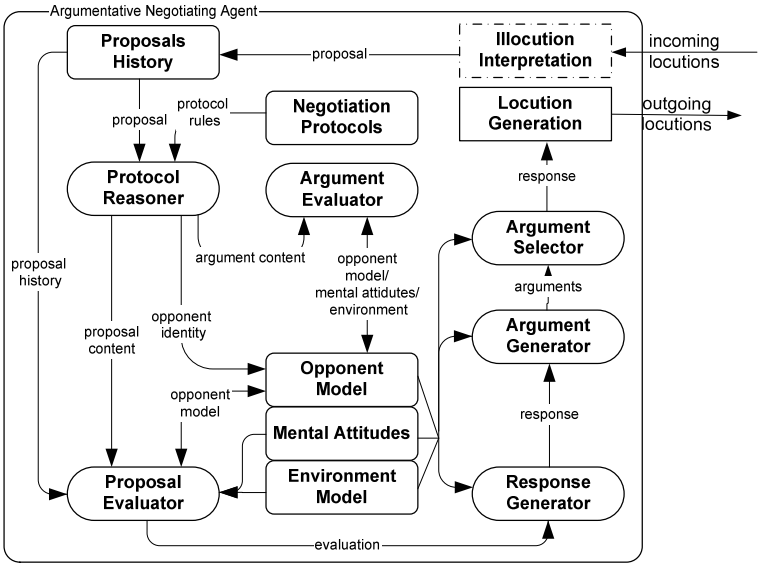

Fig. 3. Argumentation-based negotiation agent Architecture

Figure 3 shows the specification of an argumentative agent using our construction model. The figure shows how all components from the basic negotiating agent have been used, and complemented by the additional capabilities needed for ABN. We point out that the diagram has been simplified for clarity (e.g. a bidirectional link stand for a pair of unidirectional links). Furthermore, the link from *Negotiation Protocol* to *Response Generator* and *Argument Generator* has been omitted although it is, of course, necessary. Below we analyse how the descriptive and behavioural specification are affected, while with regards to the structural specification we simply point out that three new components have been added that deal with ABN.

Descriptive Specification A crucial difference between the simple negotiation agent and the ABN agent is that arguments from opponents can change the agents mental attitudes. As a result the agent's goals or motivations may change based on the new information obtained. As a result even this aspect of the descriptive specification is dynamic and the ability to refer to this changing descriptive specification directly, at run-time, by extracting the relevant attributes is crucial. The descriptive specification must also include the new decision-making capabilities of the agent.

Behavioural Specification Here the flexibility provided by the agent construction model is particularly evident. In order to deal with ABN agent we have essentially the same behavior as before (i.e. the same links and information flows), simply *extended* by links to the new controllers and refined through changes to the execution sequence. The opponent model, mental attitudes and environment model are now updated by the evaluation of the argument received before the proposal is evaluated. The response is not sent directly to the opponent but arguments are attached to the proposal by the *Argument Generator* and *Argument Selector* components. Finally, both the *Response Generator*

and *Argument Generator* use the negotiation rules in order to determine what type of responses are possible.

4 Conclusions

Negotiation, in a variety of forms, will play an increasingly more important role in the design of agent-based applications. In this paper we take the first steps towards placing negotiation within the wider context of engineering agent-based software systems...

References

1. L. Amgoud, S. Parsons, and N. Maudet. Arguments, dialogue, and negotiation. In W. Horn, editor, *Proc. ECAI 2000*, pages 338–342. IOS Press, 2000.
2. C. Bartolini, C. Preist, and N. R. Jennings. Architecting for reuse: A software framework for automated negotiation. In *Proc. 3rd Int Workshop on Agent-Oriented Software Engineering*, pages 87–98, 2002.
3. M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge. A Formal Specification of dMARS. In M. Singh, A. Rao, and M. Wooldridge, editors, *Intelligent Agents IV: Proc. ATAL 1996*, volume 1365 of *LNAI*, pages 155–176. Springer, 1996.
4. M. d'Inverno and M. Luck. Engineering AgentSpeak(L): A Formal Computational Model. *Journal of Logic and Computation*, 8(3):233–260, 1998.
5. M. d'Inverno and M. Luck. *Understanding Agent Systems*. Springer-Verlag, 2001.
6. P. Faratin. *Automated Service Negotiation Between Autonomous Computational Agents*. PhD thesis, University of London, Queen Mary and Westfield College, Dept. of Electronic Engineering, 2000.
7. S. Fatima, M. Wooldridge, and N. R. Jennings. Multi-issue negotiation under time constraints. In C. Castelfranchi and L. Johnson, editors, *Proc. AAMAS-2002*, pages 143–150. ACM Press, 2002.
8. N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge. Automated negotiation: prospects, methods and challenges. *Intt. Journal of Group Decision and Negotiation*, 10(2):199–215, 2001.
9. N. R. Jennings, S. Parsons, P. Noriega, and C. Sierra. On argumentation-based negotiation. In *Proc. of the Int. Workshop on Multi-Agent Systems*, pages 1–7, Boston, USA, 1998.
10. R. Kowalczyk. On negotiation as a distributed fuzzy constraint satisfaction problem. In *Proc. 3rd Int. Symposium on Soft Computing for Industry*, World Automation Congress, pages 631–637, 2000.
11. S. Parsons, C. Sierra, and N. Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.
12. H. Prakken and G. Vreeswijk. Logics for defeasible argumentation. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 4, pages 219–318. Kluwer, 2nd edition, 2002.
13. D. N. Walton and E. C. W. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. SUNY Press, Albany, NY, USA, 1995.