Computational Intelligence, Volume 000, Number 000, 0000

KEMNAD: A Knowledge Engineering Methodology for Negotiating Agent Development

Xudong Luo, Chunyan Miao
School of Computer Engineering
Nanyang Technological University
Singapore

Nicholas R. Jennings
School of Electronics and Computer Science
University of Southampton
Southampton, UK

Minghua He
School of Engineering and Applied Science
Aston University
Birmingham, UK

Zhiqi Shen
School of Electronical and Electronic Engineering
Nanyang Technological University
Singapore

Minjie Zhang
School of Computer Science and Software Engineering
University of Wollongong
Wollongong, Australia

Automated negotiation is widely applied in various domains. However, the development of such systems is a complex knowledge and software engineering task. So, a methodology there will be helpful. Unfortunately, none of existing methodologies can offer sufficient, detailed support for such system development. To remove this limitation, this paper develops a new methodology made up of: (1) a generic framework (architectural pattern) for the main task, and (2) a library of modular and reusable design pattern (templates) of subtasks. Thus, it is much easier to build a negotiating agent by assembling these standardised components rather than reinventing the wheel each time. Moreover, since these patterns are identified from a wide variety of existing negotiating agents (especially high impact ones), they can also improve the quality of the final systems developed. In addition, our methodology reveals what types of domain knowledge need to be input into the negotiating agents. This in turn provides a basis for developing techniques to acquire the domain knowledge from human users. This is important because negotiation agents act faithfully on the behalf of their human users and thus the relevant domain knowledge must be acquired from the human users. Finally, our methodology is validated with one high impact system.

Key words: automated negotiation, agents, knowledge engineering, software engineering, e-business.

1. INTRODUCTION

Negotiation is a communication process by which a group of entities try to reach an agreement on some matter according to the Oxford English Dictionary and the Business Dictionary (Collin, 2001). It is a subject that has been extensively discussed in game-theoretic, economic, and management science literature for decades (e.g. Nash, 1950; Raiffa, 1982; Rubinstein, 1982; Fisher *et al.*, 1991;

Hoffman *et al.*, 1994; Egels-Zandèn, 2009). Recently, there has been a surge of interest in automated negotiation systems that are populated with artificial agents (He *et al.*, 2003b; Lomuscio *et al.*, 2003; Rahwan *et al.*, 2004; Kersten and Lai, 2007; Wellman *et al.*, 2007). This is because automated negotiation can be applied into a wide range of complex computational systems such as service-oriented computing (Cappiello *et al.*, 2007; Koumoutsos and Thramboulidis, 2009), the Grid (Chao *et al.*, 2006; Guan *et al.*, 2008), peer-to-peer systems (Koulouris *et al.*, 2007; Ragone *et al.*, 2008), pervasive computing (Bagüés *et al.*, 2008; Park and Yang, 2008) and e-business (He *et al.*, 2003b; Loutaa *et al.*, 2008). Actually, it has been argued that such a negotiation is the standard mode of interaction in all systems composed of autonomous agents (Jennings, 2001). Thus, a wide range of automated negotiation systems have been developed; these include systems for auctions, direct one-to-one negotiations, and argumentation-based encounters.

However, existing negotiating agent systems are typically hand-crafted from scratch and it is a complex task. This clearly limits the potential of automated negotiation in practical applications. So, we need a practical methodology that can facilitate such developments. Unfortunately, although many agent-oriented software engineering methodologies are developed (e.g. Iglesias *et al.*, 1996; Wooldridge *et al.*, 2000; Caire *et al.*, 2001; Bresciani *et al.*, 2004; Zambonelli *et al.*, 2003; Bauer and Odell, 2005; Shen *et al.*, 2006; Oluyomi *et al.*, 2008; Beydoun *et al.*, 2009), none of them can offer sufficient, specific support for the development of negotiating agents (see Section 6 for a detailed discussion).

Against this background, in this paper we propose a novel development methodology that is able to aid the designers to model and specify automated negotiation systems. More specifically, it reduces the complex task of building an automated negotiation system to three relatively simple subtasks: situation analysis, decision making and decision executing, which the developer of a negotiating agent can focus on more easily. It then uses a generic main task model (architectural pattern) to assemble these subtasks together. After the three subtask components are substituted to the generic main task model, a relatively simple and clear task model for a negotiating agent is built up. Besides, the methodology also includes a number of design patterns (templates) for such subtasks. This can further aid the designer in the process of specifying a negotiating agent by reusing these templates.

Our methodology also identifies what types of domain knowledge are required for effective negotiation and solves the problem of how to specify these types of domain knowledge. In most cases, agents negotiate on behalf of their owner (which might be an individual or an organisation). For this to be effective, agents must be able to adequately represent their owners' interests, preferences and prejudices (*i.e.*, various types of domain knowledge) in the given domain such that they can negotiate faithfully on their behalf (Preist, 2001; Luo *et al.*, 2003b, 2004, 2006; Ludwig, 2008). However, little attention has been given to the problems of what types of domain knowledge the owner of a negotiating agent needs to impart to the agent to achieve high fidelity negotiation behaviour, nor the problems of how such types of domain knowledge are to be specified, nor how such domain knowledge can be effectively acquired from the owner. These are serious shortcomings that need to be addressed if negotiating agents are to be widely used. The methodology proposed in this paper partially solves the first two problems. As to the third problem, on the basis of the proposed methodology, we aim to further develop tools and techniques to facilitate the process of domain knowledge acquisition.

The development of the proposed methodology is based on analysing the knowledge requirements of a wide range of existing negotiating agent models, especially high impact ones (e.g. Parsons *et al.*, 1998; Kowalczyk and Bui, 2000; Barbuceanu and Lo, 2001; Faratin *et al.*, 2002; He and Jennings, 2002; Luo *et al.*, 2003a; He *et al.*, 2003a, 2006; Cheng *et al.*, 2006; Skylogiannis *et al.*, 2007). The negotiation models we selected for analysis are typical ones which cover several main classes of negotiation models, including the three basic classes of bargaining (one-to-one), auctions, and argumentation. Moreover, the models we selected are well-defined and can manifest the main characteristics of negotiation models in these three basic classes. We have also analysed the knowledge requirements used in human manual negotiation that is discussed in classic work (Pruitt, 1981; Raiffa, 1982; Fisher *et al.*, 1991; Unt, 1999). We mainly study manual negotiation subjects in the business domain, because we believe the largest class of potential users of automated negotiation systems come from this community. Therefore, in order to match our automated negotiation with manual negotiation (*i.e.*, automated negotiation can act as a manual negotiation according to humans'

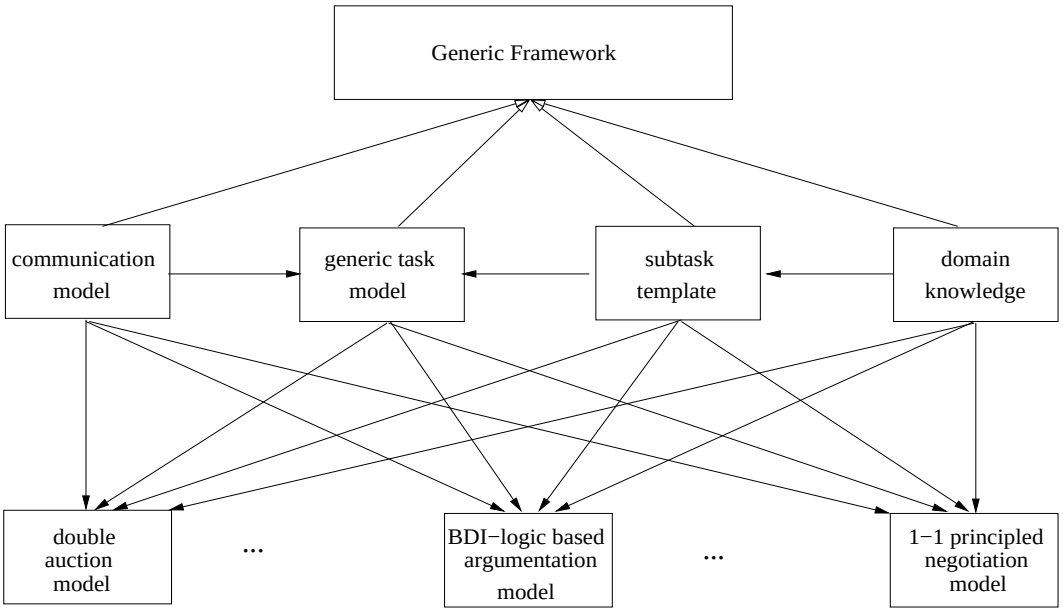


Figure 1. The overview of the presented methodology.

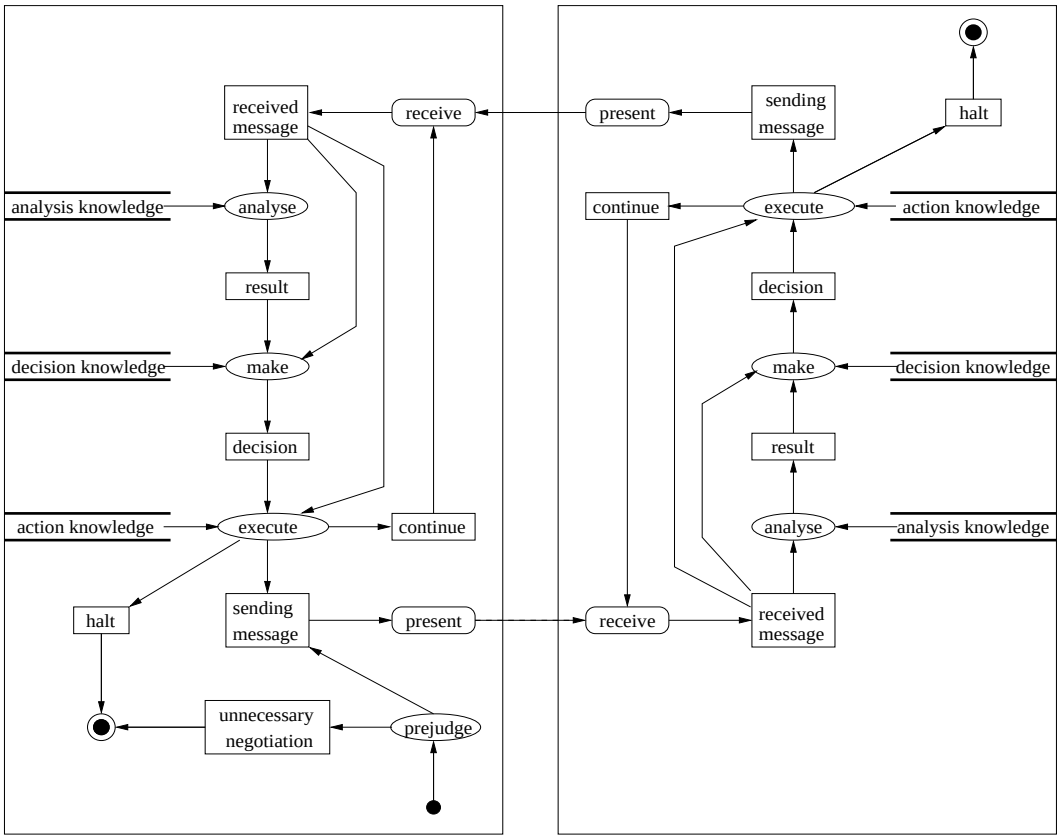
preference and knowledge), besides the analysis at the dimension of existing automated negotiation models, it is also necessary to analyse manual negotiations.

The aim of the analysis is to identify the knowledge components that are common to all (or at least many) negotiation models and those that are specific to particular types of negotiation model. In order to capture these knowledge requirements we employ the CommonKADS methodology (Schreiber *et al.*, 2000). This methodology has a long history of being exploited in knowledge intensive systems and a negotiating agent is certainly within those realms. It is possible that some extensions and variations of CommonKADS like MAS-CommonKADS (Iglesias *et al.*, 1996) can be employed here. However, CommonKADS is validated by many more applications, and the other extensions and variations have little special ingredients that we need here. So, we choose to build our work on the much more solid foundation, that is, CommonKADS.

The overview of the presented methodology is shown in Figure 1. According to the principles of knowledge engineering (Schreiber *et al.*, 2000), when building a knowledge model, one first needs to describe the generic structure of knowledge, then instantiate this and finally validate the knowledge model. Following this approach, this paper first introduces a generic knowledge model for automated negotiation systems in Section 2. Next, Sections 3 and 4 give a number of templates of the generic model's components, which can be used to instantiate the generic model. Section 5 uses an existing agent negotiation system to show the validity of our methodology. Section 6 discusses how our work advances the state-of-art in the area of agent-oriented software engineering. Finally, Section 7 summarises the paper and points out directions for further study.

2. GENERIC FRAMEWORK FOR AUTOMATED NEGOTIATIONS

A generic main task model for various automated negotiation models is presented in this section. The main task model mainly consists of two subtasks. We identify a number of templates for these subtasks. The details of these templates will be given in the following two sections, but in this section we justify why these templates are necessary for automated negotiation systems. This section also discusses various domain knowledge types that can be used in negotiations. In addition, we describe various possible types of communication between agents in negotiations. This is also one component that constitutes the main task model of automated negotiation systems.



(a) Initiator agent's main task. (b) Respondent agent's main task.

FIGURE 2. The graphical specification of the main task knowledge model of negotiation agents.

2.1. Main Task

Since negotiation can be regarded as a process by which a group of entities communicate with one another in order to reach mutually acceptable agreement on some matter (Peasall, 2001), after a negotiation is initiated, each participant always repeats the cycle that starts with receiving the other party's message, then generates a response to the received message, and finally ends with sending the response back to the other party. Generally, when generating a response to a received message, the recipient agent must consider that the generated response should advance the process towards an agreement with which he can be satisfied as much as possible. Therefore, during each cycle of the negotiation procedure, a negotiation agent faces a decision problem: given a received message, how to generate a response that moves towards an agreement which can maximise his interest? The general procedure to solve a decision problem can be divided roughly into three phases: firstly analyse the information obtained, then make a decision according to the analysis result, and finally execute the decision.

Accordingly, the main task model of a negotiation procedure can be regarded as a continual and interactive decision making procedure, each cycle of which consists of subtask analyse, make and execute (as shown in Figure 2). Subtask analyse is used to analyse the other party's message; then subtask make is used to make a decision according to the result of the analysis; and finally subtask execute is mainly used to compose an appreciate response according to the decision.

The initiator agent's main task model (as shown in Figure 2(a)) is different from that of its negotiation other party (as shown in Figure 2(b)) in the following aspect: the initiator agent starts with determining whether the negotiation is needed at all, while the respondent agent starts with the received message from the initiator agent. In particular, when a respondent agent first time receives the message from an initiator agent, the respondent agent may need to decide whether it wishes to

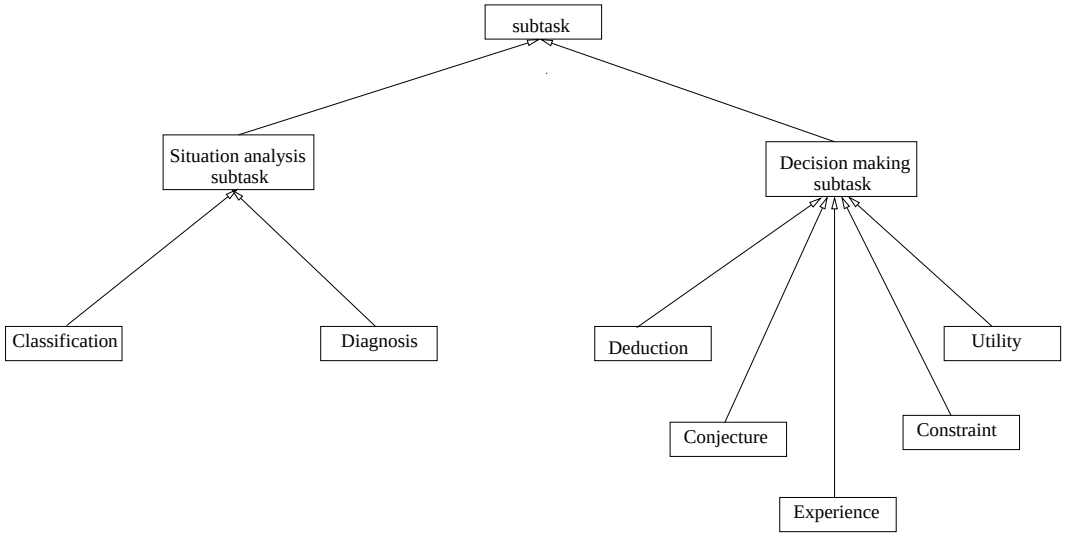


Figure 3. The overview of the templates of subtask knowledge model.

enter into the negotiation or not. However, this process can still be captured by subtasks analyse, make and execute. In fact, subtask analyse can identify the received message as an one unnecessary to be negotiated, then subtask make makes decision that it needs to go no further, and finally subtask execute composes a message to tell its negotiation partner this fact and terminates its negotiation procedure. So, it is unnecessary to put a *prejudge* subtask there. The graphical specifications of their task models are shown in Figure 2.

2.2. Templates of Subtask Knowledge Model

From the previous subsection, we know that the generic model of main task is made up of a number of subtasks (see Figure 2). Many different negotiating agents can be cast in the generic model of this main task, but they may have different subtask models. In this paper, we identify a number of templates for subtask analyse (called situation analysis) and a number of templates for subtask make (called decision making) (see Figure 3). The templates for the situation analysis subtask include the classification method and the diagnosis method; the templates for the decision making subtask include the deduction method, the conjecture method, the experience method, the constraint method and the utility method. These templates are the computational realisations of some common methods used in manual negotiations.

The classification method is one way to analyse the other party's message. In real life, when preparing a negotiation, the human negotiator often needs to identify some rules that specify what class of actions he should take given various responses of the other party (Unt, 1999). In other words, he needs classify the other party's response into different decision categories first, then make decision and respond accordingly. For example, these rules can specify when the negotiators should argue, or make a counter offer, or explain (that is responsibility of decision making subtask to further decide what the argument, or the counter offer, or the explanation is). Since automated negotiation systems simulate human negotiations to some extent, we employ the classification method in our methodology for automated negotiation.

The diagnosis method is a common method to explore the interest and motivation behind the position the other party stands on. If the interest is known then it is possible to find win-win solutions in negotiations. During the course of a negotiation, what negotiating parties often present to each other are the conflicting positions regarding the negotiation issues. However, one party is going to negotiate with another party because it needs the other to satisfy its need (Raiffa, 1982) or its interest (Fisher *et al.*, 1991) behind the position. Often there are several ways to satisfy the need/interest. Moreover, among these ways there often exists one that can satisfy both parties' needs/interests (that is, a win-win solution). The problem is how to find the win-win solution. An obvious method

is to locate the one another's need/interest behind theirs positions. The diagnosis method is one such. For example, a Chinese couple in the UK negotiate for their holiday. Their initial positions are that the wife wants to go back to China for the holiday but the husband wants to travel in Europe. Clearly, their positions are in conflict. Then, they ask each other: "why do you stand on that position?" The wife says she wants to enjoy the Chinese food and buy some Chinese stuff, and the husband says he wants to enjoy the European culture. Then they find a win-win solution: they can travel in Europe but go the cities that have China town and they will have dinner and shopping there. What the diagnosis method can do is to diagnose the interests behind conflicting positions. Clearly, the diagnosis method is different from the classification method. In fact, the classification method just directly classifies the other party's offer into a category, while the diagnosis method starts with the other party's offer to explore the interest or needs behind the offer.

We will now consider the general characteristics of the templates of the decision making subtask. In the case of the deduction method, an agent must perform two functions: perceive changes in the environment (including other agents) where it is situated, and perform actions that affect changes in the environment. Here the problem is how to choose a proper action given the changes in the environment. Hayes-Roth argues that agents reason during the course of action selection (see Murch and Johson, 1999, page 10). McCauley-Bell (1999) also holds the similar view that agents deduce their further moves from the knowledge about prior scenarios and resulting action. Moreover, Wooldridge and Parsons (2000) believe that an agent can be built as a particular type of knowledge-based system that contains a symbolic model of the environment, and that decides what actions to perform in a given situation through symbolic reasoning. Further, they illustrate this kind of agent as comprising of four components: (1) ρ: the knowledge base (typically a set of rules), (2) Δ: the logical database representing the current situation of the environment, (3) AC: the set of actions the agent can take, and (4) d: the decision algorithm, for example:

for each action a in AC do
    if $Do(a)$ can be proved from Δ using ρ then return a end-if
end-for

A typical example of reasoning employed for action selection is the BDI-logic based negotiation system developed by Parsons *et al.* (1998). In fact, during the course of a negotiation, the reasoning is used to decide how a negotiating agent should respond to its other party agent.

Let us turn to a general description of the conjecture method. During the course of a negotiation, according to the other party's offer (and the relevant information acquired from the other party), the conjecture method is often used to estimate various factors (*e.g.*, a product's reserve price (Zeng and Sycara, 1997) that determines the other party's behaviours, or the final result of a negotiation (see Ai, 2002, pages 413–502). Further, the result of the estimate can be used to decide which action should be taken in order to reach a better agreement (Zeng and Sycara, 1997; Ai, 2002). For example, according to the other party's offer price, the conjecture method proposed by Zeng and Sycara (1997) can estimate the other party's reserved price for the product to be traded, and thus determine the suitable counter offer price. Another example is the conjecture method presented in (Ai, 2002). The method can estimate the deal price according to the opening prices of both sides in a business negotiation, and thus determine suitable counter offer prices such that the estimated deal price can be agreed by the other party. Generally, the conjecture method can work well in negotiations based on two assumptions: (1) the other party's future behaviours are determined by the history of his behaviour and his current behaviour (*e.g.*, his current offer/counter-offer), and (2) the other party's behaviours will change if some other factors change.

A third template for decision making is experiential. This is based on the premise that previous negotiation experiences are useful in new negotiation (Wong *et al.*, 2000a). This assumption is reasonable. In fact, the general negotiation theory can only be used as guidelines by negotiators in different negotiation scenarios (Ai, 2002). Different negotiation parties behave differently during the course of negotiation (Unt, 1999). What the general negotiation theory reveals are common laws in different negotiations but cannot cover all specific details in various negotiations. So, in order to negotiate efficiently, negotiators must accumulate negotiation experiences and apply the experiences in practice. That is important. For example, the general business negotiation theory reveals that in a business negotiation the opening price of a buyer should be low and that of a seller should be

high. However, the general theory cannot tell how low or how high the opening price should be in individual negotiations. In this case, negotiation experience is needed. Take shopping in a market of small stores in Shanzhen city in China as an example. Actually, all prices labelled on products there are negotiable. If you just know that the opening price of buyers should be low but have no necessary experience there, you may think that using 50% of the labelled price as you opening price should be regarded as low. Nevertheless, after accumulating experiences especially through comparing your deal prices with those of local people, you will find that usually sellers' reserve prices are 30% of their labelled prices (their opening prices). Hence if your opening price is 20% of a labelled price, it is more appreciate opening "low" price for you.

A fourth method is constraint based. This is an important method for a number of reasons. *First*, in many cases negotiating parties do not know the precise details of the desired solution to the issues, and so their requirements are naturally expressed by constraints over multiple issues. For example, consider the case of an international student who just arrives in the UK for the first time and who has to rent accommodation. Since he is totally new to the country, he cannot tell a Real Estate agent exactly what he wants, but he can naturally express his requirements as constraints (*e.g.*, the accommodation should be within walking distance of the university, the rent should not be too high, and it would be better if there was an Internet connection). *Second*, negotiating parties' preferences about tradeoffs between different attributes of the desired solution can easily be modelled by fuzzy constraints. For example, although the student wants to rent accommodation near to the university, if the accommodation is really cheap, even though it is not within walking distance, he can also accept the accommodation to some extent (depending on how far and how cheap). This can be encoded as a fuzzy constraint over the combination of distance and rental, and for each combination the student can assign a satisfaction degree. *Third*, for a single solution attribute, a negotiating party might prefer certain values over others (*e.g.*, for accommodation type, the student prefers "single room in a flat" over "shared room in a house"). Such a preference can be expressed as a fuzzy constraint over a single attribute, and the preference level at a certain value of the attribute is the constraint's satisfaction degree for that value. Similarly, for multiple attributes, a buyer might prefer certain combinations of values over others (*e.g.*, for rental and period, the student prefers "cheap and short contracted period" over "expensive and long contracted period"). Such a preference can be expressed as a fuzzy constraint over multiple attributes, and the preference level at a certain combination value of these attributes is the constraint's satisfaction degree for the combination value.

Finally, we present a template for the utility based method because utility theory (Neumann and Morgenstern, 1944; Keeney and Raiffa, 1976) is a useful decision making approach in negotiations. In fact, the theory can be used to make a choice between alternative courses of action under uncertainty and risk.[1] More precisely, a utility function measures a decision maker's preference on decisions under uncertainty. Different decision makers may have different preferences and so their utility functions are different. A rational decision under uncertainty should be the action which maximises expected utility. Moreover, since uncertainty exists, the consequence of a decision may be bad, and therefore the decision maker must face a degree of risk regarding his decision. In utility theory, different utility functions can capture different decision makers' attitudes to risk, such as risk-seeking, risk-neutral and risk-aversion. In other words, the theory is suitable to deal with decision problems where risk and uncertainty are central to a decision maker's concerns. In addition, decision problems that also involve multiple issues/attributes are also suitable to be handled by utility theory. Thus, to make a proposal or a counter proposal during the course of a negotiation is a decision problem suitable to be handled by utility theory. There are three reasons for this:

(i) Often the agent has to make a counter offer under uncertainty. This is because an agent's knowledge about the other party's strategies is usually incomplete. In fact, as mentioned above, negotiating agents usually act in a self-interested fashion and so usually attempt to minimise the amount of their own private information they reveal during the encounter since any such

[1]Prospect theory (Kahneman and Tversky, 1979) and the fuzzy aggregation method (Luo and Jennings, 2007) removed some limitations of classic utility theory and can also be used in decision making under uncertainty and risk. So, we can understand them as modern utility theory. However, their delicate differences are beyond the scope of this paper and so will not be reflected in the template.

revelation will weaken their bargaining positions (Pruitt, 1981; Raiffa, 1982; Luo *et al.*, 2003a) in competitive situation.

(ii) The risk attitude of the user, whom a negotiating agent acts on behalf of, needs to be taken into account when choosing a counter proposal during the course of automated negotiation. In fact, if the user's attitude to risk is used in manual negotiations, it must be used in automated negotiation because agents must faithfully and appropriately represent their owners such that the agents will be able to negotiate competently on their behalf. Moreover, attitudes to risk often take effect in negotiations. For example, during a round of an auction, the higher a bidder agent bids, the higher the chance that the agent wins is. However, the problem is: biding too high may lead to unnecessary expense, but biding too low risks losing the chance to win. In this case, the user's risk attitude should be used to make a choice. If the user does not care about paying over the odds but cares about losing, his agent should bid high; otherwise the agent should bid low.

(iii) The problems of making proposals or counter proposals are often multi-attribute decision making problems. In fact, negotiations often need to find solutions over multiple attributes (*e.g.*, price, quality, model, volume, delivery options, expiry date, after-sale service, extended warranty options, return policy, and so on). This is because in most real negotiations what is acceptable cannot be described in terms a single parameter (Barbuceanu and Lo, 2001; Fisher *et al.*, 1991). Negotiating agents need to trade off interests where one party's gain was not necessarily the other party's loss since they have different preferences on attributes.

Having outlined the general characteristics of these methods for situation analysis and decision making, we are going to detail each in Sections 3 and 4.

2.3. Domain Knowledge

From the graphical specification of the main task knowledge model of negotiating agents (as shown in Figure 2), it can been seen that domain knowledge is used in the situation analysis subtask and the decision making subtask. In this subsection, we give an integrative picture of the various types of domain knowledge needed for the negotiation models (the detailed structure of various types of domain knowledge will be discussed together with detailed subtasks templates). For human negotiation, the following types of knowledge are often exploited by a negotiator:

(i) Objective and outcome. For example, in business negotiations, the knowledge about differences between the traded product and similar products, as well as their advantages and disadvantages are often used in bargaining. The preferences with regard to the possible outcomes of a negotiation can be used to decide a counter offer or whether an offer should be accepted.

(ii) Participant (including his experience). This type of knowledge can be used in evaluating whether the other party's offer can be acceptable. For example, in a negotiation of a student's accommodation renting scenario, sometimes although the price is quite high, if the landlord can offer an Internet connection then the student can accept because he knows he needs it for study and entertainment.

(iii) Knowledge about the other party. This type of knowledge can be used in choosing a proper negotiating strategy so that the negotiation can be carried out more efficiently. For example, in an English auction, if a bidder knows his other parties are risk averse, he needs not bid too high, and so he might win with less money.

(iv) Knowledge/preference for forming proposal or counter-proposal. The simplest form of this type of knowledge is stereotyped: if the other party's offer is like this, then the counter offer is this. The most important part of many textbooks of business negotiation is the description of knowledge about how to make (counter-)proposals.

(v) Environment. This type of knowledge includes, for example, law for business negotiations, the relationship between supply and demand in the market, and so on. This type of knowledge can be used to guarantee that (counter-)proposals (including the final agreement) are valid.

All these various types of domain knowledge need to be presented in the automated models. Take BDI-based argumentation agent (Parsons *et al.*, 1998) as an example. The agent clearly needs to know its preference for outcomes. It needs to know about its other party since it may take past

history into account in determining what arguments should be used.[2] The strategy about which argument should be used (out of the many potential alternatives) is given in type (iv) above. Other examples of this type is the knowledge that the agent uses to decide whether it should submit its strongest argument first or its weakest one. Type (v) above includes knowledge about the social context. For example, you should not, in general, send threats to your boss; if you are peers from different organisations then different types of argument may be appropriate.

These knowledge requirements can be found in most of the existing models that we have analysed. So, the generic model needs to include them. However, when a specific user employs the generic model for a specific purpose, he will tailor and customise the generic model to include the particular specification of domain knowledge types (i)–(v).

In a word, all knowledge types are used for analysing the situation of each encounter and generating the response according to the current situation.

2.4. Communication Model

Negotiating agents communicate to one another by the means of messages. In communication among humans, the intention of a message's sender is not always easily recognised. For example, "I am hot", can be regarded as an assertion, a request for a cup of cold drink, or a demand for a decrease in room temperature. However, in order to simplify the design of software agents, the message's receiver agent should have no doubt about the intention of the message's sender agent. Speech action theory (Searle, 1969) describes how humans use language to achieve everyday tasks, such as requests, orders, promises, and so on. This is employed to help define the intention of the message's sender. More precisely, the illocutionary force of this special class of utterance is used to indicate the intention of the message's sender. The *performative* in KQML (Finin *et al.*, 1994; Finin and Labrou, 1997) is used to present the intended meaning of the utterance by the sender agent.[3] Through the performative, the recipient agent can understand the intention of why the sender tells him the content of the message.

In speech action theory, illocutionary forces are classified as assertives (statements of facts), directives (commands in master-slave structure), commissives (commitments), declaratives (statements of fact), and expressive (expressions of emotion) (Huhns and Stephens, 1999). According to this classification (see Table 1) and through the analysis of human negotiation (Ai, 2002; Pruitt, 1981; Raiffa, 1982; Fisher *et al.*, 1991; Unt, 1999) as well as lots of existing automated negotiation agent systems, we identify the following performatives of messages between negotiating agents:

- `clarify`: By this performative term, the recipient agent understands that the message's content is information that the sender agent initially reveals to it. For example, in business negotiations a seller often actively provides a buyer with the product's configuration information.
- `enquire`: By this performative term, the recipient agent understands that the sender agent wants the recipient agent to provide with it some information. For example, in a business negotiation the buyer inquires the interest behind the position that the other party stands on.
- `reply`: By this performative term, the recipient agent understands that the message's content is a reply to the recipient agent's enquiry. For example, in a business negotiation the seller replies the buyer's enquiry about the interest behind the position that the other party stands on.
- `offer/counter-offer:` By this performative term, the recipient agent understands that the message's content is an offer of the sender agent. For example, in a business negotiation the seller is willing to offer the buyer the product at certain trade conditions (*e.g.*, model, quantity, delivery date, warranty, and so on).
- `request`: By this performative term, the recipient agent understands that the sender agent wants the recipient agent to do some action. For example, the seller agent asks the buyer agent to relax

[2]Such arguments can take the form of: last time I bought this product from you and we agreed that it would cost x pounds.

[3]FIPA ACL (O'Brien and Nicol, 1998) is similar to KQML in basic conception but different in some of the performatives. So, it is not important which one we choose here since we have our own performatives that are different from theirs.

TABLE 1. Performatives used in negotiation.

Illocutionary Force	**Performative**
Assertive	clarify, reply
Directive	enquire, request
Commissive	accept, reject, accept conditionally, hold firmly with argumentation, concede
Declarative	clarify, reply
Expressive	

constraints on the desired products. Actually, in different systems, the performative `request` can be instantiated as a different performative such as `relax`.

- `accept`: By this performative term, the recipient agent understands that the sender agent accepts the recipient agent's offer.
- `reject`: By this performative term, the recipient agent understands that the sender agent rejects the recipient agent's offer.
- `accept conditionally`: By this performative term, the recipient agent understands that the message's content is some condition under which the sender agent accepts the recipient agent's offer. For example, we can accept the price if you can pay fully the fee caused by something wrong with these computers during their warranty period.
- `hold firmly with argumentation`: By this performative term, the recipient agent understands that the message's content is an argumentation with which the sender agent holds its position firmly. For example, we cannot further reduce the price; otherwise we make no profit.
- `concede`: The sender agent makes a concession.

2.5. Development Processes

Basically, the development of a negotiating agent system follows the standard software engineering procedure: requirement analysis, design, coding, and test. However, as shown in Figure 4, in our methodology, the requirement analysis is carried out according to the generic main task knowledge model (standardised overall architecture pattern), then the design process becomes firstly to select the subtask knowledge models (the design patterns of negotiation situation analysis subtask and the decision making task) from the subtask knowledge template library, and to determine communication model according to the requirement analysis and the standardised communication template. These two can be viewed as architecture design of a negotiating agent. Followed is the detail design: acquire/determine the preferences on the negotiation outcomes, negotiation strategies, and even more general knowledge that is required during the course of negotiation. After the process of design is completed, all the pieces are put together to form a specification of a complete system. Since it is difficult to cover, in one paper, the details of all aspects, in this paper we focus mainly on (1) identifying the overall architecture pattern of various negotiating agents and (2) establishing the library of design pattern of components in the overall architecture.[4]

3. TEMPLATES FOR SITUATION ANALYSING SUBTASK

In the previous section, we present a generic framework for automated negotiation systems comprising of the situation analysis and decision making subtasks. We also identified a number of templates for these two subtasks, but did not give the detail of these templates. In this section we shall give the details of the situation analysis subtask templates: classification and diagnosis. These

[4]Certainly, in future work it is worth working out some guidelines and techniques to help negotiating agent developers go through the whole procedure from requirement analysis to coding and test.

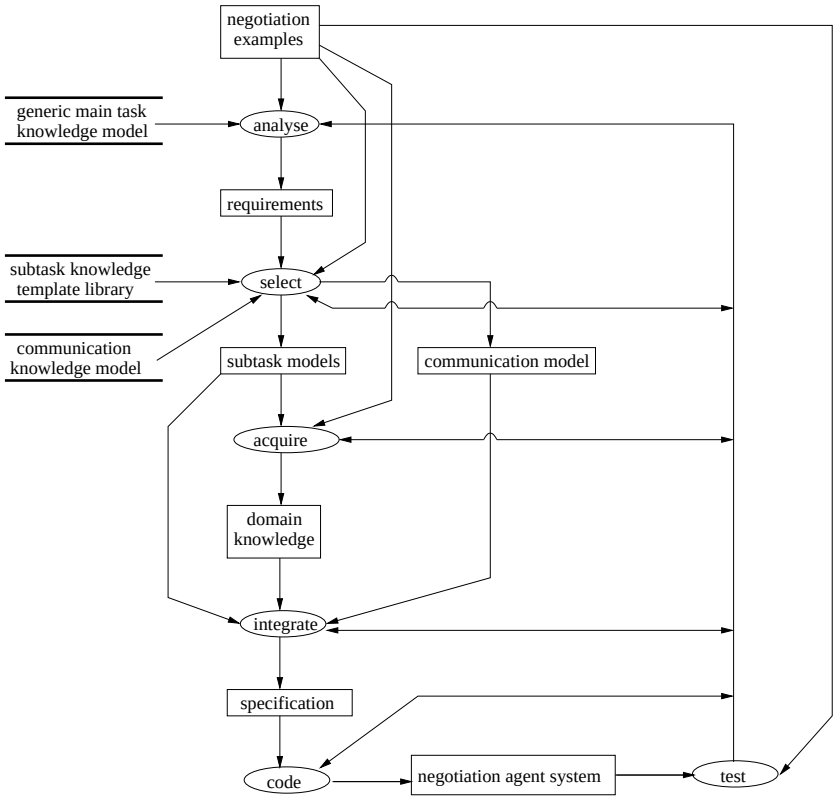


Figure 4. The lifecycle of negotiating agent development.

two templates are adapted from CommonKADS methodology (Schreiber *et al.*, 1999), but here we have tailored them suitable for use in the negotiation domain (*i.e.*, we have added and removed some components, and we also changed the terminology).

3.1. Classification

This subsection presents a template of the classification method.

3.1.1. *General Characterisation.*

Goal. Find a decision category for a piece of the other party's message based on a set of domain-specific rules.

Typical example. Find a category for a piece of the other party's message. For example, the received message is a counter offer, or a signal to terminate the negotiation, or an explanation for the previous offer.

Terminology. **Received message**: the message received from the other party, which needs to be classified. **Rules**: domain knowledge that is used in classifying the received message.

Input. The received message.

Output. A decision category.

Applicability. The method is suitable to the situation in which the information relevant to the other party's offer is sufficient to find a decision category, or the situation where the other party does not offer the relevant information.

3.1.2. *Default Method.* The basic idea of the method is: first abstract the data in the received message, then select the applicable rules, and finally judge which rule's condition is satisfied with the received message. If the condition part of a rule holds, then its conclusion part becomes a class

that the received message belongs to.[5] The textual and graphical specifications of this method are shown in Figure 5 and 6(a), respectively.

```
SUBTASK classification;
  ROLES:
    INPUT: received-message: "the message received from the other party";
    OUTPUT: situation-class: "the result of analysis";
END SUBTASK classification;

SUBTASK-METHOD classification-with-abstraction;
  REALIZES: classification;
  DECOMPOSITION:
  ROLES:
    INTERMEDIATE:
      abstracted-message: "the raw data plus the abstractions";
      rules: "a set of classification rules";
      rule: "a single classification rule";
      rule-value: "the conclusion of a rule which condition part holds";
      evaluation-results: "list of evaluated rules";
  CONTROL-STRUCTURE:
    WHILE
      HAS-SOLUTION abstract(received-message -> abstracted-message)
    DO
      received-message:=abstracted-message;
    END WHILE
    specify(abstracted-message -> rules);
    REPEAT
      select(rules -> rule);
      rules:=rules SUBTRACT rule;
      evaluate(abstracted-message + rule -> rule-value);
      situation-class:=rule-value ADD evaluation-results;
    UNTIL rules==empty END REPEAT;
END SUBTASK-METHOD classification-with-abstraction;
```

FIGURE 5. Textual specification of the rule method for classification.

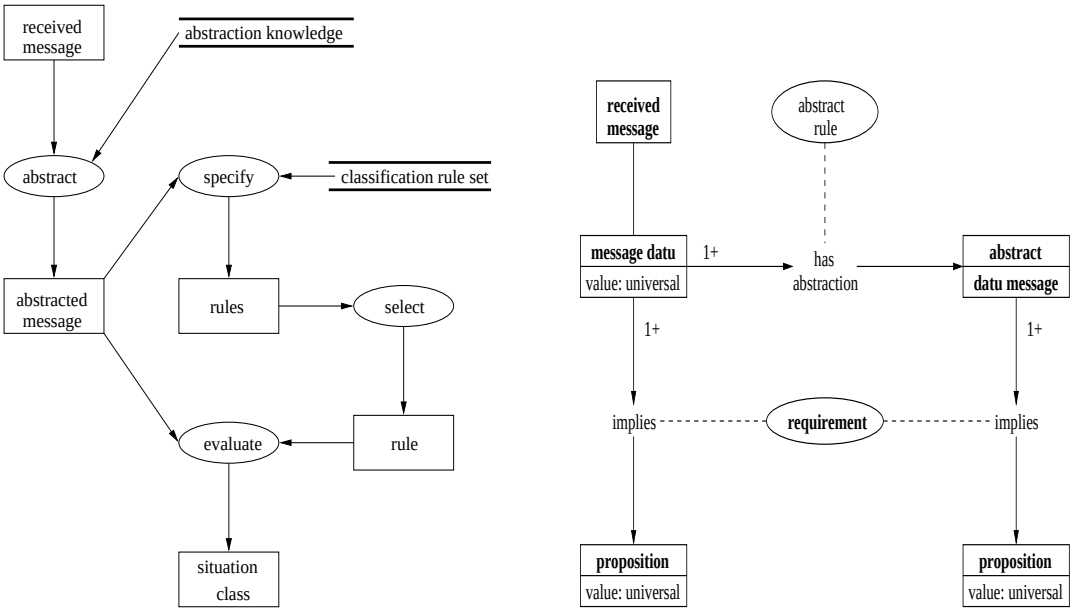
The method is built upon the following inferences:

- abstract: This inference function abstracts some of the received message. For example, in the wine selling scenario the age of the buyer might need to be abstracted, *e.g.*, any individual over 18 years old is abstracted to *adult*. The abstractions required are determined by the data used in the rules (see further). Abstraction is modelled here as an inference that is repeated until no more abstractions can be made. The abstracted features are added to the case description.
- specify: This inference finds the rules that can be applied to the abstracted message. In most classification task, the rules used are at least partially dependent on the received message, *i.e.*, the message thus plays an input role for this inference. An example of a rule in a situation classification of a continual double auction would be "if the outstanding bid is smaller than outstanding ask, and the outstanding ask is not greater than the reference price, then it is situation 1".
- select: This inference function selects one rule, from the set of rules, for evaluation.
- evaluate: This inference function checks whether the condition part of a selected rule is true or not, *e.g.*, the result of checking the above rule is "it is false that the outstanding bid is smaller than outstanding ask, and the outstanding ask is not greater than the reference price". This function is usually quite a straightforward computation.

3.1.3. *Method Variations.* In some negotiation scenarios, abstraction might not be needed, and therefore can be omitted (see Section 5 for an example). However, sometimes the data contained in the received message needs to be processed in some other way.

Sometimes, the task of a negotiation could be very simple, only a price needs to be negotiated. In this case, the specify inference always selects all the rules in the knowledge base. In other words, the specify inference is not really necessary and thus can be removed. In this case, the classification method degenerates into a simple application of a rule set.

[5]Different conclusions correspond different classes.



(a) Inference structure (b) Typical domain schema

Figure 6. Graphical specification of the method for classification.

3.1.4. *Typical Domain Schema.* An overview of this classification domain schema is given in Figure 6(b). There are three main information and knowledge types used in the default method:

(i) Specification of received message: this is a combination of the message's performative and content.
(ii) Message abstraction knowledge: it specifies dependencies between message's data (see the has-abstract rule type in Figure 6(b)).
(iii) Classification knowledge: it is represented as dependencies between the received/abstracted message and positions which combination defines different situation-class. It means that if the received/abstracted message is like this, then the rules are selected.

3.2. Diagnosis

This subsection presents a template of the diagnosis method.

3.2.1. *General Characterisation.*

Goal. Find the interest that results in the other party's position.
Typical example. The negotiation of some issues in a computer wholesale scenario.
Terminology. **Position**: the position is the solution that the other party proposes to a certain issue being negotiated. **Interest**: the cause which results in the negotiation other party's position for a certain negotiation issue. **Interest-candidates**: the set of possible interests that results in the same position. **Evidence**: acquired information that supports interest candidates.
Input. Position.
Output. Interest(s) plus the evidence acquired for supporting the hypothesis about interest(s).
Applicability. This method is suitable for a user who needs to explore the interests and goals behind the other party's position, and thus may find a solution which can satisfy both sides of the negotiation.

3.2.2. *Default Method.* The method assumes a causal network which consists of causal links between positions and interests. The network also contains causal links that indicate typical evidence for some node (see the domain schema in Figure 10).

```
SUBTASK diagnosis;
  ROLES:
    INPUT:
      position: "the negotiation other party's offer to a certain issue";
    OUTPUT:
      interest: "the interest that results in the position";
      evidence: "the evidence acquired during diagnosis";
END SUBTASK diagnosis;

SUBTASK-METHOD causal-covering;
  REALIZES: diagnosis;
  DECOMPOSITIONS:
    INFERENCE: cover, select, specify, verify;
    TRANSFER-FUNCTIONS: present, receive;
  ROLES:
    INTERMEDIATE:
      differential: "active interest candidate";
      interest-candidates: "a set of interests that could result in the position";
      result: "Boolean indicating result of the test";
      expected-evidence: "the answer acquired from negotiation other party";
   CONTROL-STRUCTURE:
     specify(position -> question);
     present(question);
     receive(interest);
     IF interest==NIL THEN
       WHILE NEW-SOLUTION cover(position -> interest-candidates) DO
         differential:=interest-candidates ADD differential;
       END WHILE
       REPEAT
         select(differential -> interest-candidates);
         specify(interest-candidates -> question);
         present(question);
         received(answer);
         evidence:=answer ADD evidence;
         FOR-EACH interest-candidates IN differential DO
           verify(interest-candidates + evidence -> result);
           IF result==false THEN differential:=differential SUBTRACT hypothesis;
           END IF;
      UNTIL
        SIZE differential =< 1 OR "no more questions left";
      END REPEAT
      interest:=differential;
    END IF;
END SUBTASK-METHOD causal-covering;
```

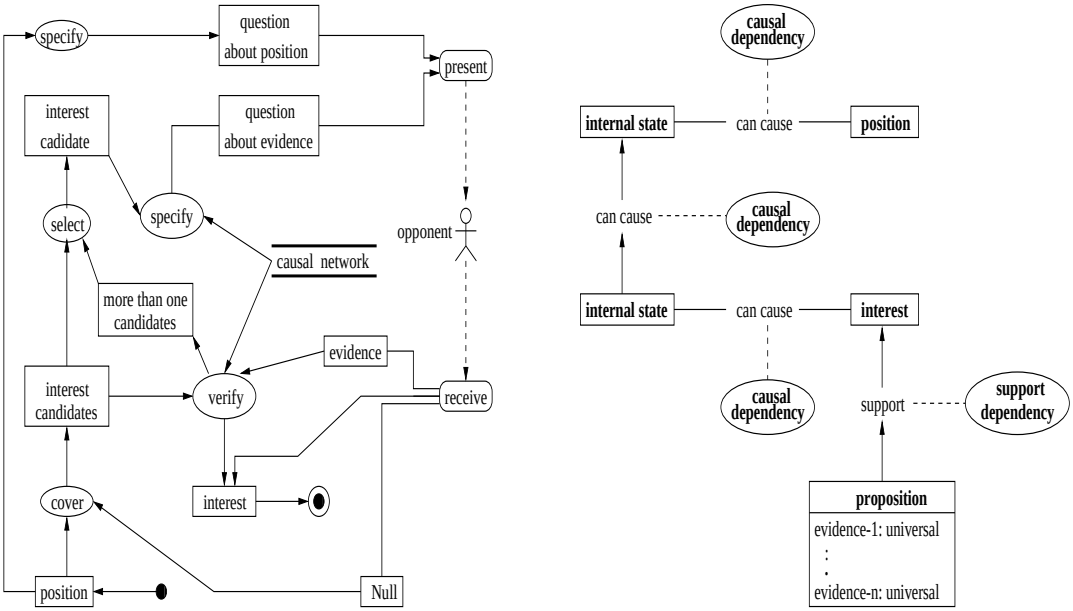
FIGURE 7. Textual specification of default causal-covering method for diagnosis.

The basic idea behind the method is: after knowing the other party's position with regard to a negotiation issue, firstly ask the other party directly what its interest behind the position. If the other party answers the question, the diagnosis procedure stops. In the case that the other party does not answer the question (or equivalently the answer is null), the agent uses its own knowledge to find all possible interests and objectives behind the position, and then asks the other party for information to determine the real interest behind the position. The textual and graphical specifications of this method are shown in Figures 7 and 8(a), respectively.

The method is built upon the following inferences and two transfer functions:

- cover: This inference searches backwards through a causal network to find all possible causes (interest candidates) behind a position. The set of interest candidates is stored in **differential**.
- select: This inference selects one interest candidate from **differential**.
- specify: This inference specifies the question of whether some evidence (related to the selected interest candidate) exists, or the question of what is the other party's interest behind his position.
- present: This transfer function presents the question to the other party.
- receive: This transfer function reads in the answer to the question.
- verify: This inference checks, according to the answer, whether an interest candidate is likely to be a real interest behind a position. If the candidate does not conflict with the answer, it is be kept in **differential**; otherwise, it is removed from **differential**.

The last four functions are executed in a loop in which the interest candidates are examined in



(a) Inference structure (b) Typical domain schema

Figure 8. Graphical specification of default causal-covering for diagnosis.

the order dictated by the select inference. The loop terminates either when the differential contains at most one interest candidate or when no more questions can be specified. Thus, the method can lead to three situations:

(i) In the case that **differential** is empty, no interests are identified. This implies that the evidence acquired from the other party is inconsistent with all the possible interests behind the position upon the negotiation issue. Or, of course, that the causal network is incomplete or in some other way different.
(ii) In the case that **differential** contains only one Candidate, the candidate is viewed as the real interest behind the position.
(iii) In the case that there are a number of candidates left in the **differential**. This implies that there are multiple interests behind the position at the negotiation issue.

3.2.3. *Method Variations.* The CommonKADS framework has identified a variety of diagnostic methods. Benjamins (1993) provided a particularly complete set of such methods. The default method presented here is a variation of one of the methods developed by Benjamins. In the following, we give a few common and relatively simple extensions and variations to the diagnostic method, which could be used in diagnostic-style situation analysis in negotiation applications.

- **Verification through explanation or consistency**. The verify inference requires either that all answers need to fully support the interest candidates, or that the answers only need to be consistent with the interest candidates. The latter option is the commonest, because in many negotiation scenarios it is difficult to acquire, from the other party, the sufficient information to explain the other party's offer. Mixed forms are also possible, *e.g.*, by stating that only a subset of the evidence needs to be explained. One usually requires that at least the initial position is explained.
- **Abstraction of answer**. It is necessary to add an inference to find an abstraction of the acquired answers when the knowledge about interests is expressed in abstract terminology that does not relate directly to the answers.
- **Multiple interests** and **interest selection**. The default method assumes that there is only one interest behind a position. This assumption can be removed by inserting the select inference after

the verify step. The select inference function picks up the most plausible one from the remaining interest candidates. A number of preference techniques have been developed for this.

3.2.4. *Typical Domain Schema.* A typical domain schema for simple diagnosis is shown in Figure 8(b). It assumes that each position is caused by some internal state, the internal state is caused by other internal state, and finally the other internal state is caused by some interest. Further, the interest is supported by the evidence that could be acquired from the negotiation with the other party.

4. TEMPLATES FOR DECISION MAKING SUBTASK

In the previous section, we gave the details of the templates of the situation analysis subtask of the generic negotiating agent. Now in this section we shall give the details of the decision making subtask templates: deduction, conjecture, experience, constraint, and utility. Notice that since the decision executing subtasks (see Figure 2) corresponding to these templates are as simple as single inference functions, we integrate them into these decision making templates' description.

Before presenting the decision making templates, we explain some terms that are shared by these templates. **Offer/counter-offer**: a negotiating agent's offer to the negotiation issue(s), for which the agent needs to determine the counter-offer and decide whether the negotiation needs to continue. **Sending-message**: the message that the agent is going to send to the other party. The message could contain the counter-offer. **Received-message**: the message that the agent has received from the other party. The message could contain the offer. **Halting-condition**: the indicator of whether the negotiation should continue or not.

4.1. Deduction

This subsection presents a template for the deduction method. This method uses strategic knowledge to deduce a counter-offer from an other party's offer and the result of its analysis subtask. The effect of this method depends mainly on the strategic knowledge.

4.1.1. *General Characterisation.*

Goal. Given a result from the situation analysis subtask and an other party's offer (stored in the received message), the method generates a counter-offer (stored in the sending message) and decides whether the negotiation should stop.

Typical example. A negotiation through reasoning and arguing (e.g. Parsons *et al.*, 1998; He *et al.*, 2003a; Skylogiannis *et al.*, 2007).

Terminology. **Analysis-result**: the decision category plus the relevant information. **Strategic Knowledge**: the domain knowledge that is used for deducing the counter-offer candidates. **Preference** and **preference ordering knowledge**: the domain knowledge that is used for choosing a counter-offer from a set of counter-offer candidates.

Input. It is the situation class and the other party's offer.

Output. It is the decision about which counter-offer should be presented to the other party and whether the negotiation should continue or not.

Applicability. The method is particularly suitable to handle the situation where the negotiating agent can obtain not only the other party's offer but also the information relevant to the offer.

4.1.2. *Default Method.* The basic idea behind the method is: firstly the negotiating agent selects applicable strategic knowledge according to the situation analysis and the other party's offer, then deduces all possible counter-offers by using the strategic knowledge, and finally picks up the most plausible one from the counter-offer candidates. The textual and graphical specifications of the method are shown in Figures 9 and 10(a), respectively.

The method is built upon the following inferences:

- select: According to the situation analysis result and the other party's offer (stored in the received

```
SUBTASK deduction;
  ROLES:
    INPUT:
      received-message: "the other party's offer to the negotiation issues";
      situation-class: "the analysis result of the other party's offer";
    OUTPUT:
      sending-message: "the message being sent to the other party";
      halting-condition: "the halting indicator of the negotiation";
END SUBTASK predication;

SUBTASK-METHOD strategically-deduce-counter-offer;
  REALIZES: deduction;
  DECOMPOSITION:
    INFERENCES: select, deduce, choose, set;
  ROLES:
    INTERMEDIATE:
      knowledge: "the knowledge selected for deducing counter-offers";
      counter-offer-candidates: "the set of possible counter-offers";
      decision: "the agent's counter-offer to the other party's offer";
  CONTROL-STRUCTURE:
    select(received-message + situation-class -> knowledge);
    deduce(knowledge + received-message + situation-class
                                  -> counter-offer-candidates);
    choose(counter-offer-candidates -> decision);
    set(counter-offer -> sending-message + halting-condition);
END SUBTASK-METHOD strategically-deduce-counter-offer;
```

Figure 9. Textual specification of the deduction method.

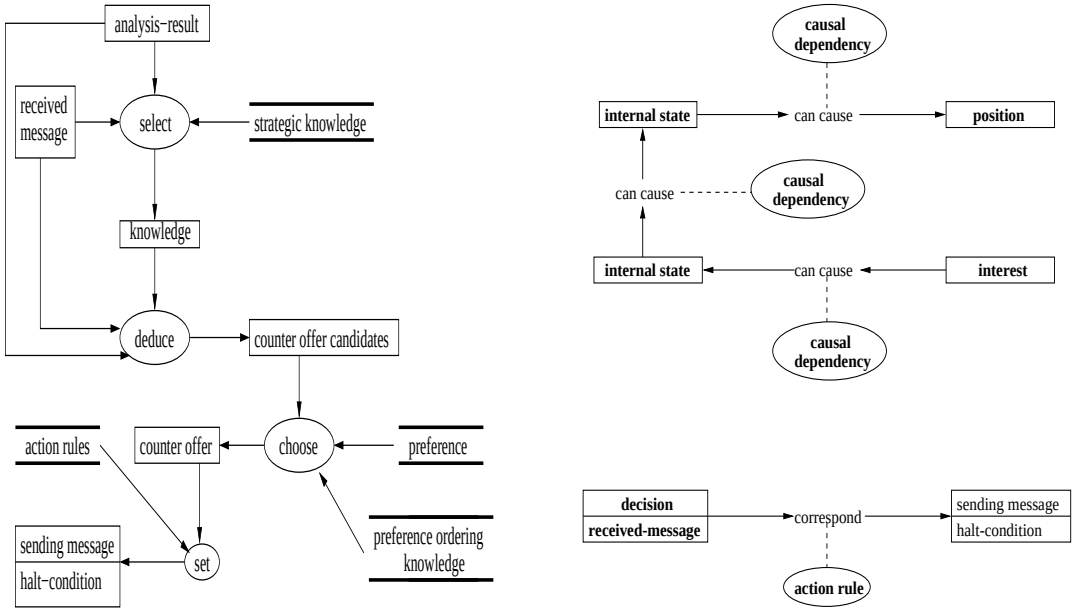
message), this inference function selects the applicable knowledge from the strategic knowledge base.

- deduce: By using the selected knowledge, this inference function deduces the counter-offer candidates from the other party's offer and its analysis result. There exit a large number of methods for implementing this inference function. A simple way that is often employed in negotiations is the forward reasoning method. For example, when the other party's interest behind its negotiation position is identified, the forward reasoning can be employed to find all alternative counter-offers (positions) which can guarantee the interest.
- choose: According to the user's preference and preference ordering knowledge, this inference function picks up a counter-offer from the set of counter-offer candidates.
- set: According to the user's action rules, this inference function executes the decision to compose the message to be sent and assign a Boolean value to the halting condition. For example, if the content of the decision is 'null' and the content of the other party's offer is also null, then the content of the sending message contains nothing, its performative is '`terminate`', and the halting condition is set to '`true`'. Actually, the inference function is the decision-executing subtask in the generic model of main task in Figure 2.

4.1.3. *Typical Domain Schema.* A typical domain schema for simple deduction is shown in Figure 10(b). It assumes that each interest can be satisfied with a number of alternative positions. Positions and interests are linked through internal states. In the simplest case, an interest can be linked directly with a position. These links constitute a causal model where the starting points and the ending points in the causal networks represent interests and positions, respectively. The causal model is used by the deduce inference function.

It assumes that a user's action knowledge is in the form of rules. In Figure 10(b), the antecedent part of such a rule is the decision made and the received message. The consequent part of such a rule is the sending message and the halting condition.

Some examples of such action rules are shown in Figure 11. The first rule says that if the decision is not null then the agent should let the other party know the decision (offer) and wait for the other party's response. The second rule says that if the decision is null and the other party cannot present any alternative offer then the agent should tell the other party that it cannot make any alternative offer, and so halt the negotiation. The third rule states that if the decision is null but the other party can make an alternative offer then the agent should ask the other party to do so.



(a) Inference structure (b) Typical domain schema

FIGURE 10. Graphical specification of the deduction method.

```
decision!='NIL'
     CORRESPOND
sending-message.offer:=decision AND
sending-message.performative:='counter-offer' AND
halting-condition:='false'

decision=='NIL' AND
received-message.performative=='no-alternative-position'
     CORRESPOND
sending-message.performative:='no-alternative-position' AND
halting-condition:='true'

decision=='NIL' AND
received-message.performative!='no-alternative-position'
     CORRESPOND
sending-message.performative:='no-alternative-position' AND
halting-condition:='false'
```

FIGURE 11. Specification of the user's action knowledge base.

4.2. Conjecture

This subsection presents a template for the conjecture method.

4.2.1. *General Characterisation.*

Goal. Given an offer of the other party, the method is used to make the counter-offer by estimating the other party's behaviour or factors that determine its behaviour.

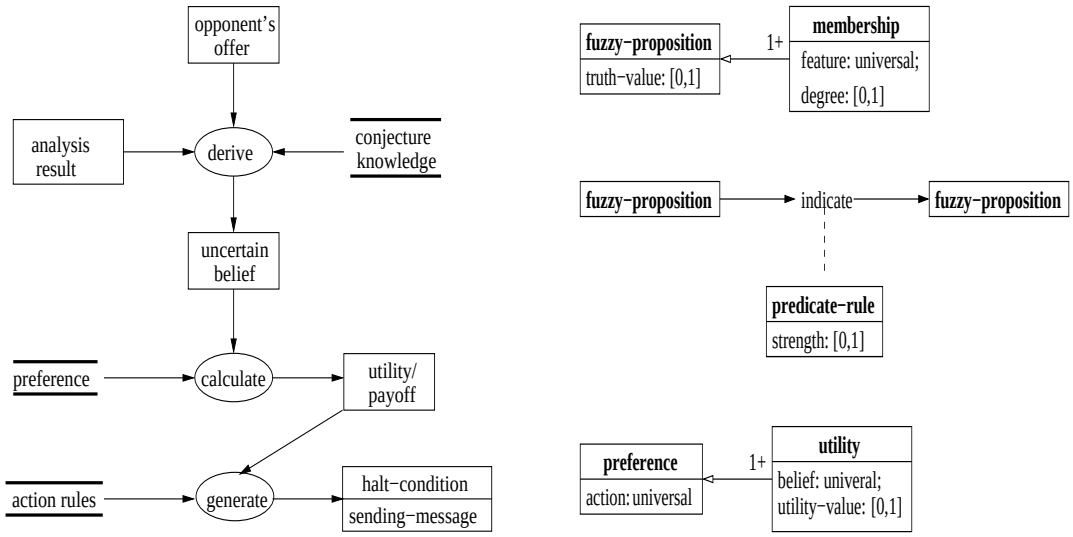
Typical example. According to the other party's offer, estimate its reservation price, and then make a counter-offer for the price (Zeng and Sycara, 1997). Another example is the trading agents (He and Jennings, 2002; He *et al.*, 2006) in which a trade agent makes a bid according to its prediction about the price tendency.

Terminology. **Situation-class**: a group of offers that share similar characteristics (*e.g.*, a group of acceptable offers). **Conjecture knowledge**: the domain knowledge that is used for building up uncertain belief on its other party's behaviours or some parameters that the other party has used to determine its offer. **Preference**: the domain knowledge about the user's preference on uncertain situations given an action.

```
SUBTASK conjecture;
  ROLES:
    INPUT:
      received-message: "the other party's offer";
    OUTPUT:
      sending-message: "the message being sent to the other party";
      halting-condition: "the halting indicator of the negotiation";
END SUBTASK conjecture;

SUBTASK-METHOD estimate-and-countermeasure;
  REALIZES: conjecture;
  DECOMPOSITION:
    INFERENCES: derive, calculate, generate;
  ROLES:
    INTERMEDIATE:
       uncertain-belief: "the belief about the factors, depending
         on which the other party agent determines its offer";
       action-expected-utility: "the expected utility of taking
         each action under all possible beliefs";
  CONTROL-STRUCTURE:
     derive(other party-offer -> uncertain-belief);
     calculate(uncertain-belief -> action-utility);
     generate(action-utility -> sending-message + halting-condition);
END SUBTASK-METHOD estimate-and-countermeasure;
```

Figure 12. Textual specification of the estimate-and-countermeasure method for conjecture.



(a) Inference structure (b) Typical domain schema

Figure 13. Graphical specification of the uncertain reasoning method for conjecture.

Input. It is the other party's offer for which the counter-offer needs to be decided.
output. It is the counter-offer (sending-message) or halting-condition.
Applicability. This method is suitable to generate a counter-offer through estimating the factors that the other party uses to determine its offer.

4.2.2. *Default Method.* The basic idea of the method is: according to the other party's offer the agent uses its knowledge about the other party to estimate the parameters that the other party has used to determine its offer, and then according to the estimate the agent determines its counter-offer. The method first builds up or updates (in the light of the new offer of the other party) its beliefs regarding the parameters that the other party has used to determine its offer, and then under the condition of such an uncertainty the agent exploits the user's preference knowledge to choose an action to produce a counter-offer which maximises the expected utility/payoff for the user. The textual and graphical specifications are shown in Figures 12 and 13(a), respectively.

The method is built upon the following inferences:

- derive: From the result of the situation analysis subtask and the other party's offer, this inference function derives the uncertain belief concerning the parameters of the other party by using conjecture knowledge. Usually, this function is realised using the uncertain reasoning models such as Bayesian networks (Pearl, 1988).
- calculate: According to the user's preference, this inference function calculates the expected utility/payoff of action candidates with respect to the belief estimated.
- generate: According to the expected utility of available action candidates, this inference function uses the action knowledge to produce a counter-offer (sending-message) and assign the halting condition to be false.

4.2.3. *Typical Domain Schema.* A typical domain schema for the estimate-and-countermeasure method is shown in Figure 13(b). It assumes that the beliefs concerning the factors that the other party uses to determine its offer can be represented by fuzzy propositions. A fuzzy proposition consists of the following components: (1) a fuzzy truth value assigned to the proposition, and (2) a membership function representing its possibility of each possible value of the factor that the other party agent uses to determine its offers. A possible link between an offer of the other party and a belief is represented as a rule with a strength meaning how strong the link is. Notice that the crisp propositions are special cases of fuzzy propositions, and accordingly although other party's offers are usually represented crisp propositions we do not specially give the schema of crisp propositions in Figure 13(b).

Given an action, the user's preference to the uncertain belief is represented as a utility function that associates a value in $[0, 1]$ to each uncertain situation.

4.3. Experience

This subsection presents a template of the experience based method.

4.3.1. *General Characterisation.*

Goal. Given an offer of the other party, the method is mainly used to decide, according to the user's previous negotiation experience, which counter-offer the negotiation agent should provide the other party with.

Typical example. Price negotiation according to previous experience (e.g. Wong *et al.*, 2000b; Lee and Kwon, 2006).

Terminology. **Negotiation experience cases**: Previous negotiation experiences that are stored in a case base. A case contains, for example, information about other party agent, trading agent and item being traded; series of concessions used by both agents in previous negotiation; and information about negotiation performance. **Contextual hierarchy** is used as an organisational structure for storing cases, which enables an efficient searching through cases. **Adaptation guideline**: the agent uses the guideline to adapt concessions from previous matching experience for use in generating the current counter-offer.

Input. The other party's offer for which the counter-offer needs to be determined.

Output. The sending message and the halting condition.

Applicability. The method is suitable for the situation where a lot of successful negotiation cases are available.

4.3.2. *Default Method.* The basic idea of the method is: in the situation similar to the previous successful negotiation case, the strategy implied in the previous case is reused to determine the counter-offer.[6] The textual and graphical specifications of the method are shown in Figures 14 and 15(a), respectively.

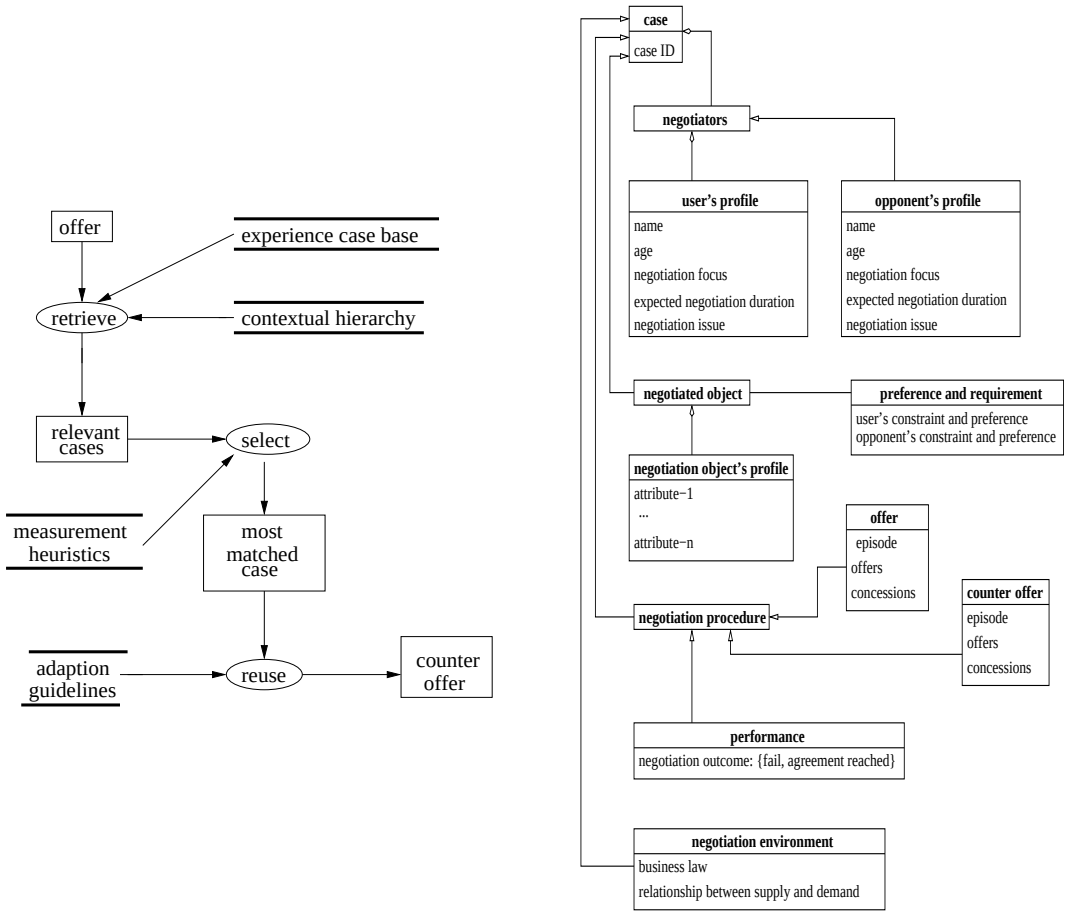
The method is built upon the following inferences:

[6]This sounds like: last time we did this and then we got success; since this time the situation is similar, we will do the same thing hoping we will get success again.

```
SUBTASK experience;
  ROLES:
    INPUT:
      received-message: "the other party's offer";
    OUTPUT:
      sending-message: "the message being sent to the other party";
      halting-condition: "the halting indicator of the negotiation";
END SUBTASK experience;

SUBTASK-METHOD case-based-reasoning;
  REALIZES: experience;
  DECOMPOSITION:
    INFERENCES: retrieve, select, reuse;
  ROLES:
    INTERMEDIATE:
      relevant-cases: "the cases that are  similar to the current situation";
      current-best-match: "the case that is the most similar to the current situation";
  CONTROL-STRUCTURE:
    retrieve(receiving-message -> relevant-cases);
    select(relevant-cases -> current-best-matched);
    reuse(most-matched-case -> sending-message + halting-condition);
END SUBTASK-METHOD case-based-reasoning;
```

Figure 14. Textual specification of the case based method for experience.



(a) Inference structure (b) Typical domain schema

Figure 15. Graphical specification of the case-based method for experience.

- retrieve: This inference function retrieves the relevant previous negotiation experience from the repository by using the organisational structure. In the field of case based reasoning, many methods exist to realise inference function retrieve.
- select: This inference function selects a most matched case.
- reuse: This inference function proposes a suitable counter-offer from the selected negotiation experience case according to the adaptation guideline. This inference function also decides whether the negotiation needs to be continued.

4.3.3. *Typical Domain Schema.* A typical domain schema for reusing previous negotiation experiences is shown in Figure 15(b). It assumes that the user's experience is represented as cases. A case may contain the following items:

(i) Negotiators.

- The user's profile that could include the user's name, age, occupation, and so on.
- The other party's profile (its content is similar to the user's profile).

(ii) Negotiated object.

- The profile of the negotiated object. For example, the profile of a second-hand car includes: engine size, mileage, year of manufacture, and so on.
- The user's requirement and preference with respect to the negotiated object, which may include negotiation focus, expected negotiation duration, negotiated issue (*e.g.*, price, warranty, trade-in, and so on), constraint (*e.g.*, budget, rental period, and so on), and preferences.

(iii) Negotiation procedure.

- Offers made by the other party and concessions used in episodic strategies.
- Counter-offers made by the negotiation participant and concessions used in episodic strategies.
- Performance information about the feedback of the negotiation result. It may include the outcome of negotiation: success or failure in reaching the final mutually agreed solution to the negotiation issues (*e.g.*, price).

(iv) Negotiation environment. It may include, for example, law in respect to business negotiation, or else the relationship between supply and demand in the market.

4.4. Constraint

This subsection presents a template for the constraint-based method.

4.4.1. *General Characterisation.*

Goal. Given an offer of the other party, the method decides the response that the negotiating agent should make.

Typical example. An accommodation renting problem in which the buyer's requirements and preferences with respect to the desired accommodation are represented as fuzzy constraints (Luo *et al.*, 2003a). Other examples include the scenarios of car trading (Kowalczyk and Bui, 2000) and supply chain (Wang *et al.*, 2009).

Terminology. **Checking message** and **relaxing message**: two kinds of received messages from the other party, for which a response message needs to be decided. **Constraint set**: Constraints express the user's requirement/preferences on the negotiated object. **User's profile**: the user's background information that might be useful for evaluating the other party's offer. **Decision**: the result of the agent's decision actions and the type of the result. According to the decision, the agent may carry out some further action, for example, ask the other party for a new proposal according to the relaxed constraint.

Input. It is the other party message with the identity.

Output. It is the response decided.

```
SUBTASK constraint;
  ROLES:
    INPUT:
      received-message: "the other party's message plus its type";
    OUTPUT:
      sending-message: "the message being sent to the other party";
      halting-condition: "the halting indicator of the negotiation";
END SUBTASK constraint;

SUBTASK-METHOD critique-or-relax;
  REALIZES: constraint;
  DECOMPOSITION:
    INFERENCES: verify, evaluate, critique, relax, generate;
  ROLES:
    GLOBAL INTERMEDIATE:
      submitted-constraints: "the constrains submitted to the other party";
    LOCAL INTERMEDIATE:
      decision: "the decision  made against the other party's offer";
  CONTROL-STRUCTURE:
    IF received-message.performative == 'check' THEN
      IF verify(received-message) == 'true' THEN
         evaluate(received-message -> decision.content);
         decision.flag:='evaluation';
      ELSE
        critique(received-message -> decision.content);
        submitted-constraints=submitted-constraints ADD decision.content;
        decision.flag:='criticism';
      END IF;
    END IF;
    IF received-message.type == 'relax' THEN
      IF HAS-SOLUTION relax(submitted-constraints -> decision.content)
        THEN decision.flag:='relaxation';
        ELSE decision.flag:='no-more';
      END IF;
    END IF;
    generate(decision -> sending-message + halting-condition);
END SUBTASK-METHOD critique-or-relax;
```

Figure 16. Textual specification of the critique-or-relax method for constraint.

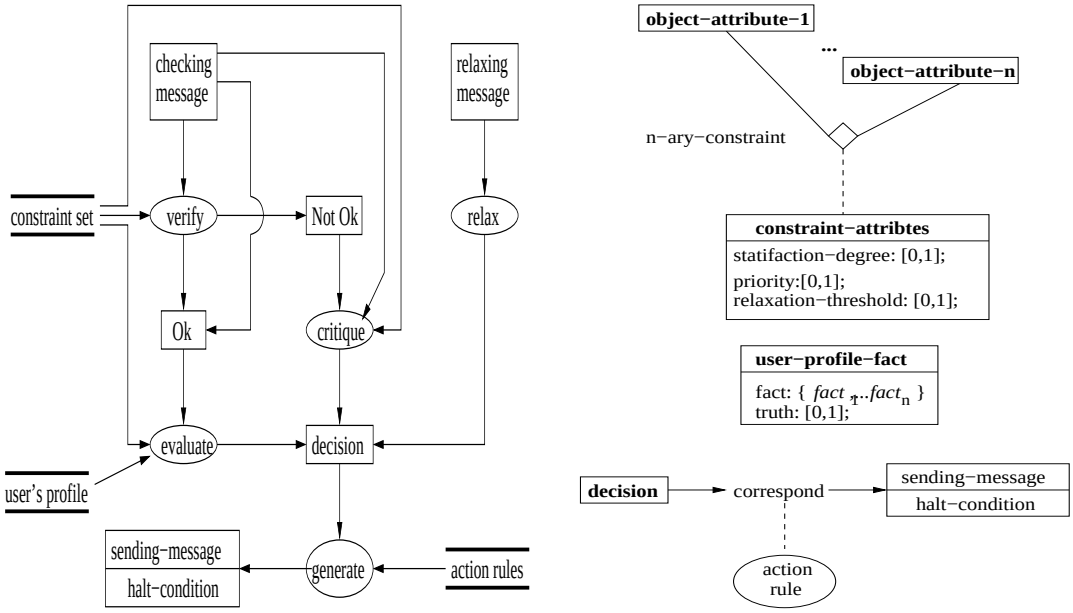
Applicability. This method is suitable to the situation where users cannot describe exactly what they want, but they can use constraints to represent their requirements and preferences on single issues as well as on the trade-off between multiple issues.

4.4.2. *Default Method.* There exist a number of methods for constraint based negotiation. Here we present the generalisation of the fuzzy constraint based method developed by Luo *et al.* (2003a). The basic idea is as follows:

(i) After receiving a checking message that contains the other party's offer, firstly the agent checks whether any of its user's constraints is violated by the other party's offer. If one constraint is violated, the agent will send the constraint to the other party and ask the other party for a new offer. Otherwise, the agent will evaluate the other party's offer under the further consideration of the user's profile.
(ii) After receiving a relaxing message, the agent will relax one submitted constraint; and if there exits one submitted constraint that can be relaxed, send back to the other party the relaxed constraint as a replacement, otherwise it will tell the other party that it cannot relax any constraint.

The textual and graphical specifications of the method are shown in Figures 16 and 17(a), respectively. The method is built upon the following inferences:

- verify: This inference function checks whether an other party's offer is consistent with the user's constraint set. If they are consistent, the inference function returns '`true`'; otherwise, it returns '`false`'.
- evaluate: This inference function calculates the user's acceptability for an other party's offer and checks whether the acceptability is greater than the acceptability threshold.



(a) Inference structure (b) Typical domain schema

FIGURE 17. Graphical specification of the critique-or-relax method for constraint.

- critique: This inference function specifies a constraint, from the constraint set, which is violated by the other party's offer.
- relax: This inference function relaxes one constraint that the agent has already submitted to the other party agent. If no submitted constraints can be relaxed, the inference function returns null (denoted as NIL); otherwise it returns the relaxed constraint.
- generate: This inference function executes the decision made to compose the sending message and assign a Boolean value to the halting condition according to the user's action knowledge base, which is usually represented by rules. For example, if the flag of its parameter is '`relaxation`', that is, the content of its parameter is obtained from inference function relax, then this inference function composes the sending message and assigns a Boolean value to the halting condition according to the following rule: if the relaxation fails, tell the other party that the negotiation has broken down and the negotiation terminates; and if the relaxation is successful, then send the other party the relaxed constraint and ask for a new offer.

4.4.3. *Typical Domain Schema.* A typical domain schema for the constraint-based method is shown in Figure 17(b). It assumes that the user's requirements and preferences concerning the object of negotiation are represented by fuzzy constraints on the object's attributes (each ranging over a finite domain). A fuzzy constraint consists of a number of the (combination of) values that the object's attributes can take. A degree is associated to each value (or the value combination) of attributes of a constraint. It represents the user's satisfaction degree with the attribute's value (or the combination of the attributes' values). Each constraint is associated with a priority indicating the importance level of the constraint among these constraints. Besides, a constraint is also associated with a relaxation threshold, meaning that the constraint can be regarded as being broken if the satisfaction degree is lower than the relaxation threshold. An example of the instance of such a constraint is shown in Figure 18.

Sometimes, the user's profile is also useful when considering whether an offer of the other party is acceptable or not. In Figure 17(b), fuzzy truth propositions introduced by Zadeh (1965, 1975) are employed to represent the facts in the user profile model because these facts are often partially true, *i.e.*, they have truth values between "completely true" and "completely false". For example, a student who wants to rent accommodation might like to be allowed to keep a pet with a truth

```
KNOWLEDGE-BASE student-requirements;
  USE: buyer-schema;

  INSTANCE distance constraint;
    INSTANCE-OF: n-ary-constraint;
    ATTRIBUTES:
      distance: distance<15-minute-walk;
      priority: 0.37;
  END INSTANCE distance constraint;

  INSTANCE period constraint;
    INSTANCE-OF: n-ary-constraint;
    ATTRIBUTES:
      rental-period: rental-period<12-months;
      priority: 0.33;
  END INSTANCE period constraint;

  INSTANCE rent-and-period-constraint;
    INSTANCE-OF: n-ary-constraint;
      ATTRIBUTES:
        priority: 0.3;
        threshold: 0.6;

    TUPLE
      ARGUMENT-1: cost<=250;
      ARGUMENT-2: NIL;
      ATTRIBUTES:
        Satisfaction-degree: 100%;
    END TUPLE

    TUPLE
      ARGUMENT-1: 250<cost<=260;
      ARGUMENT-2: NIL;
      ATTRIBUTES:
        satisfaction-degree: 90%;
    END TUPLE

    TUPLE
      ARGUMENT-1: 260<cost<=270;
      ARGUMENT-2: NIL;
      ATTRIBUTES:
        satisfaction-degree: 80%;
    END TUPLE
    TUPLE
      ARGUMENT-1: 270<cost<=280;
      ARGUMENT-2: NIL;
      ATTRIBUTES:
        satisfaction-degree: 70%;
    END TUPLE
    TUPLE
      ARGUMENT-1: 280<cost<=290;
      ARGUMENT-2: NIL;
      ATTRIBUTES:
        satisfaction-degree: 60%;
    END TUPLE

    TUPLE
      ARGUMENT-1: 290<cost<=300;
      ARGUMENT-2: NIL;
      ATTRIBUTES:
        satisfaction-degree: 40%;
    END TUPLE

    TUPLE
      ARGUMENT-1: 300<cost<=310;
      ARGUMENT-2: NIL;
      ATTRIBUTES:
        satisfaction-degree: 30%;
    END TUPLE

    TUPLE
      ARGUMENT-1: 3100<cost<=3200;
      ARGUMENT-2: NIL;
      ATTRIBUTES:
        satisfaction-degree: 20%;
    END TUPLE

    TUPLE
      ARGUMENT-1: 3200<cost<=3300;
      ARGUMENT-2: NIL;
      ATTRIBUTES:
        satisfaction-degree: 10%;
    END TUPLE;

    TUPLE
      ARGUMENT-1: rent>3300;
      ARGUMENT-2: NIL;
      ATTRIBUTES:
        satisfaction-degree: 0;
    END TUPLE

  END INSTANCE rent-and-period-constraint;

END KNOWLEDGE-BASE student-requirements;
```

Figure 18. Specification of the buyer's requirement/preference knowledge base.

value expressed as 60%, he might be more committed to having a telephone (truth value expressed as 70%), even more to an efficient heater (80%) and much less to the furniture being new (30%). Their specification is shown in Figure 19.

It assumes that a user's action knowledge is in the form of rules. In Figure 17(b), the condition part of such a rule is the decision made by the inferences evaluate, critique or relax. Actually, the *decision* contains two kinds of information: (1) the consequence of a previous inference, and (2) the flag indicating which inference makes the decision. The consequent part of such a rule is a sending message and a halting condition.

Examples of such action rules are shown in Figure 20. The first rule means that if the agent thinks an offer of the other party is acceptable, then the negotiation succeeds in reaching a deal. The second rule implies that if the agent does not think an offer of the other party is acceptable, then it should ask the other party for an alternative offer and wait for the other party's answer. The third rule states that if the agent finds a constraint that cannot be satisfied with the other party's offer, then it should ask the other party for an alternative offer that can satisfy the constraint, and wait for the other party's response. The fourth rule indicates that if the agent succeeds in relaxing a constraint, then it should let the other party know about relaxation, and wait for the other party's

```
KNOWLEDGE-BASE student-profile;
  USE: buyer-schema;

  INSTANCE sex;
    INSTANCE-OF: buyer-profile-fact;
      ATTRIBUTES:
        fact: 'female';
        truth: 1;
  END INSTANCE cooking;

  INSTANCE cooking;
    INSTANCE-OF: buyer-profile-fact;
      ATTRIBUTES:
        fact: 'cooking';
        truth: 0;
  END INSTANCE cooking;

  INSTANCE pet;
    INSTANCE-OF: buyer-profile-fact;
      ATTRIBUTES:
        fact: 'pet';
        truth: 0.6;
  END INSTANCE pet;

  INSTANCE phone;
    INSTANCE-OF: buyer-profile-fact;
      ATTRIBUTES:
        fact: 'phone';
        truth: 0.7;
  END INSTANCE phone;

  INSTANCE new-furniture;
    INSTANCE-OF: buyer-profile-fact;
      ATTRIBUTES:
        fact: 'new furniture';
        truth: 0.3;
  END INSTANCE new-furniture;

  INSTANCE air-conditioner
    INSTANCE-OF: buyer-profile-fact;
      ATTRIBUTES:
        fact: 'air conditioner';
        truth: 0.4;
  END INSTANCE air-conditioner;

END KNOWLEDGE-BASE student-profile;
```

FIGURE 19. Specification of the user's profile knowledge base.

```
decision.flag=='evaluation' AND decision.content=='true'
     CORRESPOND
sending-message.performative:='deal' AND halting-condition:='true'

decision.flag=='evaluation' AND decision.content=='false'
     CORRESPOND
sending-message.performative:='re-find' AND halting-condition:='false'

decision.flag=='criticism'
     CORRESPOND
sending-message.constraint:=decision.content
AND sending-message.performative:='find' AND halting-condition:='false'

decision.flag=='relaxation' AND decision.content!=NIL
     CORRESPOND
sending-message.constraint:=decision.content AND
sending-message.performative:='find' AND halting-condition:='false'

decision.flag=='relaxation' AND decision.content==NIL
     CORRESPOND
sending-message.performative:='fail' AND halting-condition:='false'
```

FIGURE 20. Specification of the user's action knowledge base.

new offer. The fifth rule says that if the agent cannot relax a constraint, then it should let the other party know, and then wait to see what will happen.

4.5. Utility

This subsection presents a template of the utility method.

4.5.1. *General Characterisation.*

Goal. Given an offer of the other party, the method determines an offer that maximises the expected utility.

Typical example. Business negotiations about price and quantity.

Terminology. **Utility function**: a measure that is defined in an uncertain environment for measuring a user's relative preference on options. **Utility-indicator**: the indicator representing the current level of expected utility of the counter-offer. **Alternative-offer**: a new offer that can keep unchanged the current level of the expected utility.

Input. The signal from the situation analysis subtask, which indicates whether the method should commence.

```
SUBTASK utility;
  ROLES:
    INPUT:
      received-message: "the message received from the other party";
      making-alternative-offer: "a signal that starts the process";
      offer-history: "the set of offers the negotiating agent used to provide";
    OUTPUT:
      sending-message: "the message being sent to the other party";
      halting-condition: "the halt indicator of the negotiation";
      offer-history: "the set of offers the negotiating agent used to provide";
END SUBTASK utility;

SUBTASK-METHOD tradeoff-first-concede-second;
  REALIZES: utility;
  DECOMPOSITION:
    INFERENCES: trade-off, concede, set;
  ROLES:
    GLOBAL INTERMEDIATE:
       utility-indicator: "the indicator representing the current level of
                           the expected utility of its counter-offer";
    LOCAL INTERMEDIATE:
       counter-offer: "the counter-offer that can keep the current level of
                       the expected utility unchanged";
       decision: "the generated counter-offer plus its type";
  CONTROL-STRUCTURE:
    IF making-counter-offer==‘true’ THEN
       REPEAT
         IF HAS-SOLUTION tradeoff(utility-indicator + offer-history -> alternative-offer)
           THEN decision.content:=alternative-offer; decision.flag:=‘tradeoff’;
                offer-history:=offer-history ADD alternative-offer;
           ELSE concede(utility-indicator -> utility-indicator);
                IF utility-indicator<utility-threshold THEN
                  decision.flag:=‘no-more-concede’;
                END IF
         END IF
       UNTIL
         utility-indicator<utility-threshold OR decision.flag==‘trade-off’
       END REPEAT
    END IF;
    set(decision + received-message -> sending-message + halting-condition);
END SUBTASK-METHOD tradeoff-first-concede-second;
```

Figure 21. Textual specification of the tradeoff-first-concede-second method for utility based decision making.

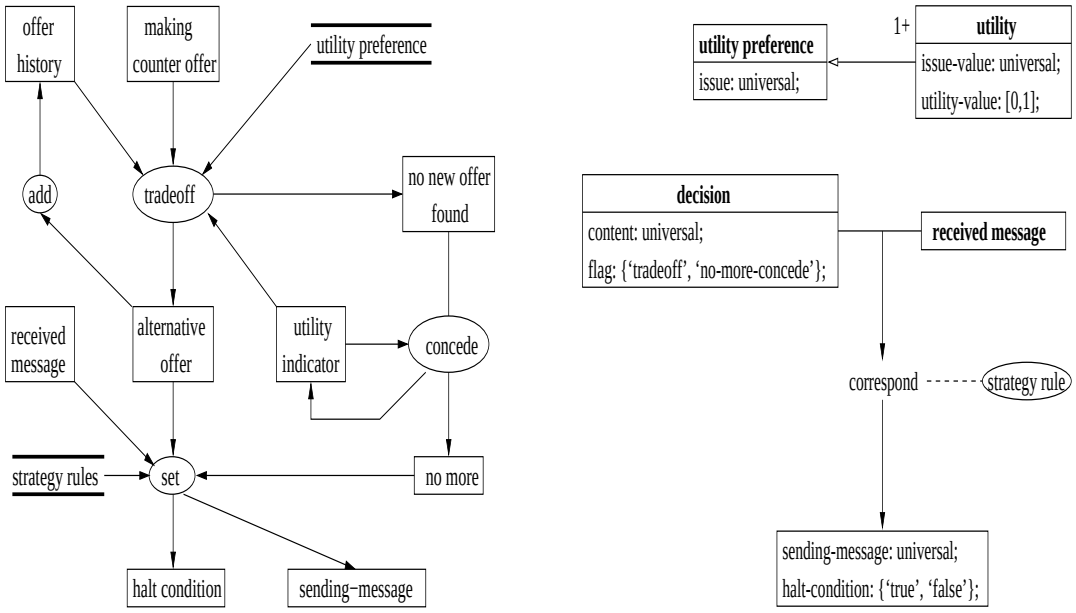
Output. The counter-offer (sending message) and halting condition.

Applicability. The utility method is suitable for the situations where the users like to use their own utility functions to assess potential offers, try to maximise their own utility assessment, and think the increase of the utility of some issue(s) can compensate for the decrease in the utility of other issue(s). Actually, a utility function can be regarded as a special form of a fuzzy constraint. Accordingly, the method can be viewed as a special treatment on this kind of fuzzy constraint. Rather, the method in the previous subsection is a general treatment on general fuzzy constraints representing he user's requirements/preferences on a negotiation object.

4.5.2. *Default Method.* The basic idea of the method is: when the other party does not accept an offer, the agent first tries to find an alternative offer that can keep its expected utility unchanged; if the agent cannot find such an alternative, it reduces its expected utility as little as possible and then finds another offer that can achieve the reduced expected utility. Its textual and graphical specifications are shown in Figures 21 and 22(a).

The method is built upon the following inferences:

- tradeoff: This inference function generates an alternative offer that can keep the current level of the expected utility unchanged. If the inference function cannot find a (new) counter-offer, it returns null (denoted as "NIL"); otherwise, it returns the alternative offer.
- concede: This inference function reduces the current level of the expected utility. The user determines how much of the reduction to make. Generally, the reduction rate is predefined by the user whom the negotiating agent acts on behalf of. That is, this inference function is realised as



(a) Inference structure (b) Typical domain schema

FIGURE 22. Graphical specification of the first-tradeoff-concede-second method for utility.

```
decision.flag==‘trade-offer’

       CORRESPOND
sending-message.offer:=decision.content AND
sending-message.performative:=‘counter-offer’ AND halting-condition:=‘false’
decision.flag==‘no-more-concede’ AND
received-message.performative==‘no-more-concede’

      CORRESPOND
sending-message.performative:=‘no-more-concede’ AND halting-condition:=‘true’
decision.flag==‘no-more-concede’ AND
received-message.performative!=‘no-more-concede’

      CORRESPOND
sending-message.performative:=‘no-more-concede’ AND halting-condition:=‘false’
```

FIGURE 23. Specification of the user's action knowledge base.

a fixed reduction. However, it could also be realised as a knowledge based reduction (specifying what reduction should be made under a certain condition).

- set: This inference function carries out the decision to compose the sending message and assign a Boolean value to the halting condition according to the user's action rules.

4.5.3. *Typical Domain Schema.* A typical domain schema for the first-tradeoff-concede-second method is shown in Figure 22(b). It assumes that the user's utility preference is represented by utility functions corresponding to each of the individual issues. A pair of issue value and utility value represents the utility of each possible solution to the issue that is negotiated.

It assumes that a user's action knowledge is in the form of rules. In Figure 22(b), the antecedent part of such a rule is the decision made by the inference tradeoff or concede. The decision contains the following information: (1) the consequence of a previous inference, and (2) flag indicating whether a solution is found. The subsequent part of such a rule is a sending message and a halting condition.

Examples of such action rules are shown in Figure 23. The first rule states that if the agent makes a tradeoff, then it should tell the other party agent what the tradeoff is, and then see what will happen. The second rule states that if the agent cannot concede any more when the other party cannot concede any more either, then it should tell the other party agent that it cannot concede

```
SUBTASK prejudge;
  ROLES:
    INPUT:
    OUTPUT:
      necessary-to-negotiate: "the necessity for negotiation";
      persuader-proposal: "the message that will be sent to the persuadee";
END SUBTASK prejudge;

SUBTASK-METHOD prejudge-through-trying;
  REALIZES: prejudge;
  DECOMPOSITION:
    INFERENCES: select, prove, build;
  ROLES:
    INTERMEDIATE:
      intention: "the objective that the persuader wants to achieve by himself or
        by persuading the persuadee to accept";
      argument: "the argument set used to persuade the persuadee to accept its
        objective or drop the objection";
  CONTROL-STRUCTURE:
    select(-> intention);
    IF prove(intention)==‘no-proof-found’
      THEN necessary-to-negotiation:=‘true’;
           build(intention -> argument);
           persuader-proposal.performative:=‘convince’;
           persuader-proposal.content:=argument;
      ELSE necessary-to-negotiation:=‘false’;
    END IF;
END SUBTASK-METHOD prejudge-through-trying;
```

Figure 24. Specification of the persuader's knowledge of prejudge subtask.

further either, and halt the negotiation. The third rule implies that if the agent cannot concede any more but the other party might be able to concede, then it should tell the other party agent that it cannot concede any more, but hope the other party can do something.

5. SPECIFICATION OF BDI-LOGIC BASED NEGOTIATION AGENT

This section will show the validity of our methodology by using a high impact negotiation system developed in (Parsons *et al.*, 1998).[7] It is a BDI-logic based negotiation system that automated the human negotiations that have the following form:

A : Please give me a nail. I need it to hang a picture.
B : I can't give you a nail because I need it to hang a mirror.
A : You may use a screw to hang the mirror.
B : Why not? OK, I give you the nail.

More specifically, this section gives the specification of the negotiating system with the above illustration by the methodology. We do this from top to bottom: (1) the task models of two negotiating agents, (2) the communication model that is used in the task models, (3) the inference function used in the task models, and (4) domain knowledge that inference functions operate on.

5.1. Task Knowledge Models

According to the generic model of the initiator agent's main task presented in Subsection 2.1, the persuader's task model consists of four subtask models (since the persuader is the initiator of the negotiation) and one main task model. The persuader agent employs subtask prejudge (as shown in Figure 24) to judge whether a negotiation is necessary or not. If it can do the thing it intends to do, no negotiation is necessary; otherwise a negotiation is necessary. Subtask persuader-situation-analysis

[7]The impact of (Parsons *et al.*, 1998) can easily be checked out through Google Scholar (http://scholar.google.co.uk). Until 7 December 2010, it is cited 610 times. Given this, we believe it is more compelling to use such a system to confirm the validity of our methodology, rather than use one that is more recently published but whose quality and impact has not been proved so strongly yet.

```
SUBTASK persuader-situation-analysis;
  ROLES:
    INPUT:
      persuadee-proposal: "the message received from the persuader";
    OUTPUT:
      situation-class: "the situation class of the currently received message";
END SUBTASK persuader-situation-analysis;

SUBTASK-METHOD assess-through-performative;
  REALIZES: persuader-situation-analysis;
  CONTROL-STRUCTURE:
    IF persuadee-proposal.performative==‘accept’ THEN situation-class:=1; END IF;
    IF persuadee-proposal.performative==‘convince’ THEN situation-class:=2; END IF;
END SUBTASK-METHOD assess-through-performative;
```

FIGURE 25. Specification of the persuader's knowledge of situation analysis subtask.

```
SUBTASK persuader-decision-making;
  ROLES:
    INPUT:
      persuadee-proposal: "the message received from the persuader";
      situation-class: "the situation class of the currently received message";
    OUTPUT:
      way: "an alternative way of achieving the original objective of the
        persuader, or a way to persuading the persuadee to drop his objection,
        or no way";
      argument: "the argument used to persuade the persuadee to drop his
        objection or accept the persuader’s objective";
END SUBTASK persuader-decision-making;

SUBTASK-METHOD decide-through-proving;
  REALIZES: decision-making;
  DECOMPOSITION:
    INFERENCES: find-way;
  CONTROL-STRUCTURE:
    IF situation==1 THEN way:=‘no-more’; END IF;
    IF situation==2 THEN
       find-way(persuadee-proposal.content -> way + argument) END IF;
END SUBTASK-METHOD decide-through-proving;
```

FIGURE 26. Specification of the persuader's knowledge of subtask decision making.

(as shown in Figure 25) is used to classify the message received from the persuadee. According to the analysis result, the persuader-decision-making subtask (as shown in Figure 26) decides the counter argument or whether to continue the negotiation. The persuader-decision-executing subtask (as shown in Figure 27) is used to compose the sending message and assign a Boolean value to the halting condition according to the decision made. Finally, main task persuasion (as shown in Figure 28) assembles the four subtasks together.

The method of persuader's situation analysis subtask (shown in Figure 25) is a simplified version of the classification method shown in Figures 5 and 6(a): inference functions abstract, specify and select there are all omitted here, and evaluate there is implemented by two "IF ... THEN" rules. Notice that the rules in the domain knowledge there are actually the condition part of the two "IF...THEN" rules.

The persuader's decision making subtask (shown in Figure 26) is a variation of the deduction method shown in Figures 9 and 10(a): inference function select there is implemented explicitly by the two "IF ... THEN ..." sentences here. In the first case, there is no deduction involved; and in the second case, the deduce inference there is implemented by the find-way inference here, and the selected strategic knowledge used by the find-way inference function are the persuader's BDI-base (shown in Figure 36), the action theory (shown in Figure 38), the persuader's planning rules (shown in Figure 39) and the bridge rules (shown in Figure 41).

In Figure 27, we give the specification of the persuader's decision executing subtask. Actually, it consists of two rules used to compose the sending message and assign a Boolean value to the halting condition according to the decision made by the persuader's decision-making subtask.

The specification of persuader's main task knowledge is shown in Figure 28. Clearly, it is an instantiation of the generic main task model shown in Figure 2. Actually, the generic main task

```
SUBTASK persuader-decision-executing;
  ROLES:
    INPUT:
      way: "an alternative way to achieve the original objective of the
        persuader, or a way to persuade the persuadee to drop his objection,
        or no way";
      argument: "the argument set used to persuade the persuadee to drop his
        objection or accept the persuader's objective";
    OUTPUT:
      persuader-proposal: "the message being sent to the persuadee";
      halting-condition: "the indicator whether the negotiation should stop";
END SUBTASK persuader-decision-executing;

SUBTASK-METHOD act-through-plan;
  REALIZES: persuader-decision-executing;
  CONTROL-STRUCTURE:
    IF way==‘alternative-way’ OR  way==‘no-way’ OR  way==‘no-more’
      THEN persuader-proposal.performative:=‘end’;
           persuader-proposal.content:=NIL; halting-condition:=‘true’;
      ELSE IF way==‘argument-way’ THEN
             persuader-proposal.performative:=‘convince’;
             persuader-proposal.content:=argument; halting-condition:=‘false’;
    END IF; END IF;
END SUBTASK-METHOD act-through-plan;
```

Figure 27. Specification of the persuader's knowledge of decision executing.

```
TASK persuasion;
  ROLES:
    OUTPUT:
      result: "the argument that is accepted by the persuadee";
END TASK persuasion;

TASK-METHOD persuade-through-argumentation;
  REALIZES: persuasion;
  DECOMPOSITION:
    SUBTASK: prejudge, persuader-situation-analysis, persuader-decision-making,
             persuader-decision-executing;
    TRANSFER-FUNCTIONS: receive, present;
  ROLES:
    INTERMEDIATE:
      necessary-to-negotiate: "the necessity for negotiation";
      situation-class: "the situation class of the currently received message";
      persuader-proposal: "the message received from the persuader";
      persuadee-proposal: "the message sent to the persuader";
      way: "an alternative way to achieve the original objective of the persuader
        agent, or a way to persuade the persuadee to drop his objection, or no way";
      argument: "the argument used to persuade the persuadee to drop his objection
        or accept the objective";
  CONTROL-STRUCTURE:
    prejudge(-> necessary-to-negotiate + persuader-proposal);
    IF necessary-to-negotiate==‘true’ THEN
      present(persuader-proposal);
      REPEAT
        receive(persuadee-proposal);
        situation-class:=persuader-situation-analysis(persuadee-proposal);
        (way,argument):=persuader-decision-making(situation, persuadee-proposal);
        (halting-condition, persuader-proposal):=persuader-decision-executing(way, argument);
        present(persuader-proposal);
      UNTIL halting-condition==‘true’ END REPEAT;
    END IF;
END TASK-METHOD persuade-through-argumentation;
```

Figure 28. Specification of the persuader's knowledge of main task.

captures some commonalities of many negotiation approaches. It reduces a complex task of building an automated negotiation system to the three relatively simple subtasks of situation analysis, decision making and decision executing, and further we have supplied many templates for such subtasks in this paper. This must facilitate the designer to give the specification of an automated negotiation system. However, perhaps programmers are not happy with the method of separating the three subtasks since sometimes doing so leads to that some parts of such specifications are redundant. In

```
TASK persuasion;
  ROLES:
    OUTPUT:
      result: "the argument that is accepted by the persuadee";
END TASK persuasion;

TASK-METHOD persuader-through-argumentation;
  REALIZES: persuasion;
  DECOMPOSITION:
    INFERENCES: select, prove, build, find-way;
    TRANSFER-FUNCTIONS: receive, present;
  ROLES:
    INTERMEDIATE:
      intention: "the objective that the persuader wants to achieve by itself or
        by persuading the persuadee to accept";
      way: "an alternative way of achieving the original objective of the persuader,
        or a way to persuading the persuadee to drop his objection, or no way";
      argument: "the argument set used to persuade the persuadee to drop his
        objection or accept the persuader's objective";
      persuader-proposal: "the message received from the persuader";
      persuadee-proposal: "the message sent to the persuader";
 CONTROL-STRUCTURE:
    select(-> intention);
    IF prove(intention)==‘no-proof-found’ THEN
       build(intention -> argument);
       persuader-proposal.performative:=‘convince’;
       persuader-proposal.content:=argument;
       present(persuader-proposal);
       REPEAT
         receive(persuadee-proposal);
         IF persuadee-proposal.performative!=‘accept’ THEN
           find-way(persuadee-proposal.content -> way + argument);
           IF way==‘alternative-way’ OR  way==‘no-way’ THEN
             persuader-proposal.performative:=‘end’;
             persuader-proposal.content:=NIL;
           ELSE IF way==‘argument-way’ THEN
             persuader-proposal.performative:=‘convince’;
             persuader-proposal.content:=argument;
           END IF; END IF;
           present(persuader-proposal);
         END IF;
       UNTIL
         way==‘alternative-way’ OR way==‘no-way’ OR
         persuadee-proposal.performative==‘accept’
       END REPEAT;
    END IF;
END TASK-METHOD persuade-through-argumentation;
```

FIGURE 29. Integrated specification of the persuader's knowledge of task.

this case, the three subtasks can be substituted to the generic main task, and thus after simplifying we can have a clearer version of the task model of a negotiating agent. For example, we can substitute the persuader's subtasks into its main task model (shown in Figure 28) and after simplifying we get the one as shown in Figure 29.

Now we turn to the persuadee's task model. Similarly, we can give the persuadee's situation analysis, decision making, and decision executing subtask models, but for the sake of simplicity and space, we just give out its final integrated task model.

Finally, we put the graphic specification of the persuader and persuadee agents' task knowledge together in Figure 31. From the figure, we see clearly the negotiation procedures of these two agents and their interaction as well as the domain knowledge.

5.2. Communication Knowledge

The specification of the messages between the persuader and the persuadee is shown in Figure 32. A message the persuader sends to the persuadee is represented as concept **persuader-proposal** which has attributes **performative** and **content**. When the performative takes value '`convince`', it means that the persuader agent wants to convince the persuadee using the message; when the performative takes value '`end`', it means that the persuader agent wants to terminate the negotia-

```
TASK objection;
  ROLES:
    OUTPUT: result: "the argument that is accepted by the persuadee";
END TASK objection;

TASK-METHOD object-through-argumentation;
  REALIZES: objection;
  DECOMPOSITION:
    INFERENCES: examine, vet;
    TRANSFER-FUNCTIONS: receive, present;
  ROLES:
    INTERMEDIATE:
      persuader-proposal: "the message received from the persuader";
      arguments: "the argument set in which each argument attacks the persuader's argument";
      persuadee-proposal: "the message sent to the persuader";
  CONTROL-STRUCTURE:
    receive(persuader-proposal);
    WHILE examine(persuader-proposal)!=NIL OR persuader-proposal.performative!='end' DO
      evaluate(persuader-proposal.content -> arguments);
      vet(arguments -> persuadee-proposal.content);
      persuadee-proposal.performative:='argue'; present(persuadee-proposal);
      receive(persuader-proposal);
    END WHILE;
    IF examine(persuader-proposal)==NIL THEN
      persuadee-proposal.performative:='accept'; persuadee-proposal.content:=NIL;
      present(persuadee-proposal);
      result:=persuader-proposal.content;
    END IF;
    IF persuader-proposal.performative=='end' THEN result:='not convince' END IF;
END TASK-METHOD object-through-argumentation;
```

Figure 30. Integrated specification of the persuadee's knowledge of task.

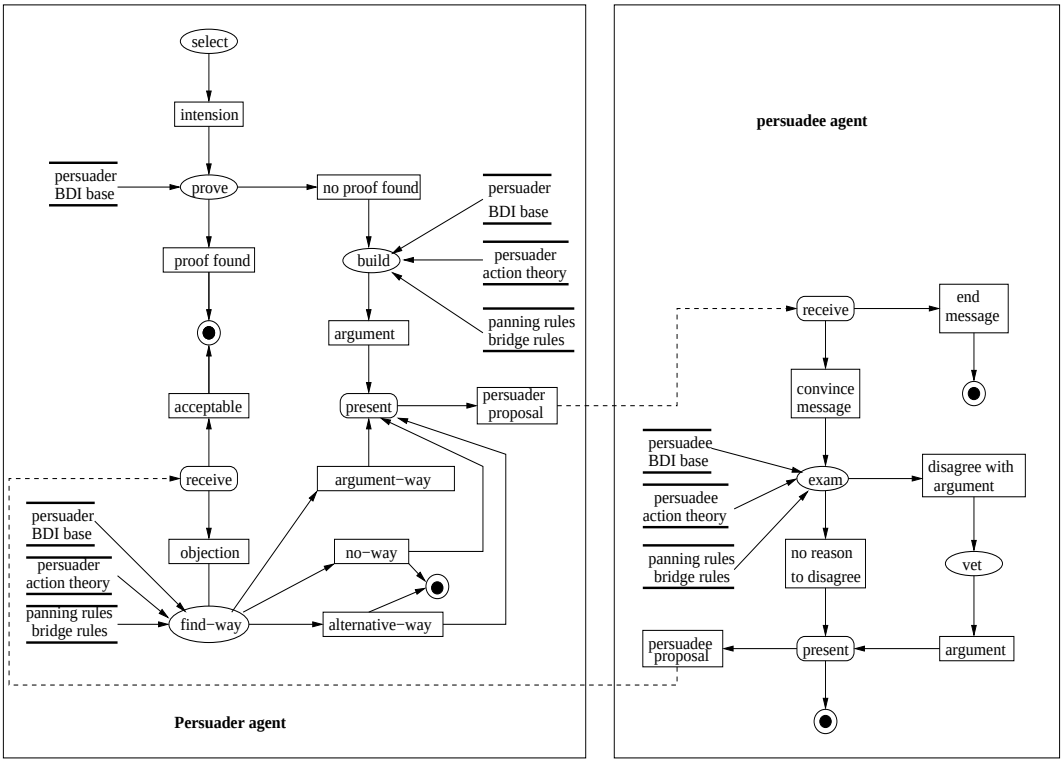
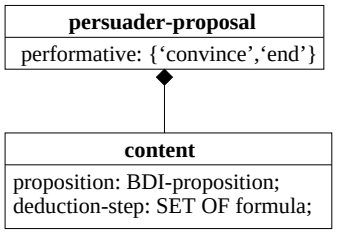


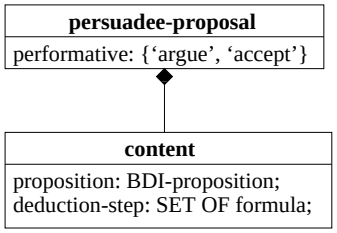
Figure 31. Graphic specification of task knowledge of the persuader and persuadee agents.

tion. The content contains a BDI-proposition and the deduction step of the BDI-proposition. The deduction step is comprised of a set of formulas. A formula is made up of a prerequisite (a set of

```
MESSAGE persuader-proposal;
  DESCRIPTION:
    "The format of a message the persuader
     agent sends to the persuadee agent.";
  PERFORMATIVE: {'convince', 'end'};
  CONTENT:
    proposition: BDI-proposition;
    deductive-step: SET-OF formula;
END MESSAGE persuader-proposal;
```




```
MESSAGE persuadee-proposal;
  DESCRIPTION:
    "The format of a message the persuadee
     agent sends to the persuader agent.";
  PERFORMATIVE: {'argue','accept'};
  CONTENT:
    proposition: BDI-proposition;
    deductive-steps: SET-OF formula;
END MESSAGE persuader-proposal;
```




```
CONCEPT formula;
  DESCRIPTION:
    "A formula is a deductive step built
     using only inference rules, bridge
     rules, planning rules, action rules and
     the BDI base of the corresponding agent.";
  ATTRIBUTES:
     prerequisite: SET-OF BDI-proposition;
     consequence: BDI-proposition;
END CONCEPT formula;
```

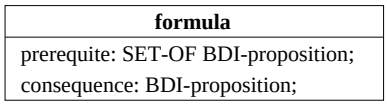


FIGURE 32. Specification of communication knowledge between persuader and persuadee agents.

BDI-proposition) and a consequence (a BDI-proposition). Similarly, concept **persuadee-proposal** can be understood.

5.3. Inference Knowledge

The specification of the inference knowledge of the persuader and persuadee agents are shown in Figure 33. In total, six inference functions are employed by the persuader and persuadee agents. In particular, inference functions select, prove, build and find-way are used by the persuader agent, while other two inference functions examine and vet are used by the persuadee.

- select: This inference selects an intention to achieve. It has (1) one dynamic knowledge role—output role **intention** which is played by domain concept **BDI-proposition**, and (2) one static knowledge role **BDI-base** which is also played by domain concept **BDI-proposition**.
- prove: This inference function finds a proof for an intention by using the agent's BDI-base. It has (1) two dynamic knowledge roles: input role **intention** and output role **result**, and (2) one static knowledge role: **BDI-base**. These three knowledge roles are all played by domain concept **BDI-proposition**.
- build: This inference function generates an argument for an intention. It has (1) two dynamic knowledge roles: input role **intention** and output role **argument**); and (2) one static knowledge role: **BDI-base**. These three knowledge roles are all played by domain concept **BDI-proposition**.
- find-way: This inference function finds an alternative way to achieve the original intention of the persuader, or a way to persuade the persuadee to drop his objection; if the either way cannot be found, return 'NIL'. It has (1) three dynamic knowledge roles: input role **objection** and output roles **way** and **argument**, and (2) one static knowledge role: **BDI-base**. Knowledge roles **objection**, **argument** and **BDI-base** are played by domain concept **BDI-proposition**. Knowledge role **way** takes value on {'alternative-way', 'no-way', 'argument-way', 'no-more'}. When **way** takes value 'alternative-way', it means that the persuader finds an alternative way to gain its objective; when **way** takes value 'no-way', it means that the persuader cannot find any argument to persuade the persuadee further; when **way** takes value 'argument-way', it means that

```
INFERENCE select;
  ROLES:
    INPUT:
    OUTPUT: intention;
    STATIC: BDI-base;
  SPECIFICATION:
    "To select an intention to be satisfied.";
END INFERENCE select;

KNOWLEDGE-ROLE intention;
  TYPE: DYNAMIC;
  DOMAIN-MAPPING: BDI-proposition;
END KNOWLEDGE-ROLE intention;

KNOWLEDGE-ROLE BDI-base;
  TYPE: STATIC;
  DOMAIN-MAPPING: BDI-proposition;
END KNOWLEDGE-ROLE BDI-base;

INFERENCE prove;
  ROLES:
    INPUT: intention;
    OUTPUT: result:{'proof found',
                    'no proof found'};
    STATIC: BDI-base;
  SPECIFICATION:
    "To find a proof for an intention based
     on its BDI base. If impossible, return
     'NIL'.";
END INFERENCE prove;

INFERENCE build;
  ROLES:
    INPUT: intention;
    OUTPUT: argument;
    STATIC: BDI-base;
  SPECIFICATION:
    "To generate an argument for an intention.";
END INFERENCE build;

KNOWLEDGE-ROLE argument;
  TYPE: DYNAMIC;
  DOMAIN-MAPPING: BDI-proposition;
END KNOWLEDGE-ROLE argument;

INFERENCE find-way;
  ROLES:
    INPUT: objection;
    OUTPUT: way, argument;
    STATIC: BDI-base;
  SPECIFICATION:
    "To find an alternative way to achieve the
     original objective of the persuader, or a
     way to persuade the persuadee to drop his
     objection. If the either way cannot be
     found, return 'NIL'.";
END INFERENCE find-way;

KNOWLEDGE-ROLE objection;
  TYPE: DYNAMIC;
  DOMAIN-MAPPING: BDI-proposition;
END KNOWLEDGE-ROLE objection;

KNOWLEDGE-ROLE way;
  TYPE: DYNAMIC;
  DOMAIN-MAPPING: {'alternative-way',
    'no-way', 'argument-way', 'no-more'};
END KNOWLEDGE-ROLE way;

INFERENCE examine;
  ROLES:
    INPUT: proposal;
    OUTPUT: flag: {'no-reason-to-disagree',
                   'disagree-with-argument'};
            arguments;
    STATIC: BDI-base;
  SPECIFICATION:
    "To see whether the persuadee should
     agrees with the persuader's proposal";
END INFERENCE examine;

KNOWLEDGE-ROLE proposal;
  TYPE: DYNAMIC;
  DOMAIN-MAPPING: BDI-proposition;
END KNOWLEDGE-ROLE proposal;

KNOWLEDGE-ROLE arguments;
  TYPE: DYNAMIC;
  DOMAIN-MAPPING: SET-OF BDI-proposition;
END KNOWLEDGE-ROLE argument;

INFERENCE vet;
  ROLES:
    INPUT: arguments;
    OUTPUT: argument;
  SPECIFICATION:
    "To pick up the the most acceptable
     argument from a set of arguments.";
END INFERENCE critique;
```

Figure 33. Specification of the inference knowledge for persuader and persuadee agents.

the persuader has found an argument to persuade the persuadee further; when **way** takes value 'no-more', it means that the persuadee has already accepted the persuader's proposal and so no more negotiation is needed.

- examine: This inference function checks whether the persuadee should agree with the persuader's proposal. It has (1) three dynamic knowledge roles: input role **proposal** which is played by domain concept **BDI-proposition**, and two output roles **flag** (which just takes value on {'no-reason-to-disagree', 'disagree-with-argument'}) and **argument** (which is played by domain concept **BDI-proposition**); and (2) one static knowledge role **BDI-base** which is played by domain concept **BDI-proposition**.

```
CONCEPT can;
  DESCRIPTION:
    "A description of an ability of an agent.";
 ATTRIBUTES:
   agent-name: STRING;
   can-do-thing: STRING;
END CONCEPT can;

CONCEPT have;
  DESCRIPTION:
    "A description of things that an agent have.";
  ATTRIBUTES:
    owner-name: STRING;
    thing: STRING;
END CONCEPT have;

CONCEPT give;
  DESCRIPTION:
    "A description of the action an agent
     gives a thing to another agent.";
  ATTRIBUTES:
    giver-name: STRING;
    receiver-name: STRING;
    thing: STRING;
END CONCEPT give;

CONCEPT ask;
  DESCRIPTION:
    "A description of the action an agent
     asks another agent for a thing.";
ATTRIBUTES:
  asker-name: STRING;
  responder-name: STRING;
  thing: STRING;
END CONCEPT ask;

CONCEPT tell;
  DESCRIPTION:
    "A description of the action an agent
     tells another agent what can be given.";
  ATTRIBUTES:
    teller-name: STRING;
    hearer-name: STRING;
    thing: STRING;
END CONCEPT tell;

CONCEPT BDI-proposition;
  DESCRIPTION:
    "A description of a proposition with
     modal Belief, Desire and Intention.";
  ATTRIBUTES:
    proposition: {can, have, give, ask,
                  tell, BDI-proposition},
      CARDINALITY: 1+;
    prefix: {'Belief', 'Desire',
             'Intention', 'NOT'};
END CONCEPT BDI-proposition;
```

can
agent−name: STRING; can−do−thing: STRING;

have
owner−name: STRING; thing: STRING;

give
giver: STRING; receiver: STRING;

ask
asker−name: STRING; responder−name: STRING; thing: STRING;

tell
teller−name: STRING; hearer−name: STRING; thing: STRING;

BDI−proposition
proposition: {can, give, ask tell, BDI−proposition}; prefix; {'Belief', 'Desire', 'Intention', 'NOT'};

FIGURE 34. Concept schema of the domain knowledge of persuader and persuadee agents.

- vet: This inference function picks up the most acceptable argument from a set of arguments.[8] This inference function has two dynamic knowledge roles: input role **argument** and output role **argument** both of which are played by domain concept **BDI-proposition**.

5.4. Domain Knowledge

In the system, there are two negotiating agents *persuader* and *persuadee*. There are seven domain concepts used by both agents, whose specification is shown in Figure 34. Concept **can** is used to

[8]The definition is that an argument is more acceptable than another one can be found just before Subsection 4.3 of (Parsons *et al.*, 1998).

```
RULE-TYPE ability-rule;
  ANTECEDENT: BDI-proposition;
  CONSEQUENT: can;
  CONNECTION-SYMBOL: indicates;
  ATTRIBUTE:
    Modal: {'Belief', 'Desire',
              'Intention', 'NOT'};
END RULE-TYPE ability-rule;

RULE-TYPE ownership-rule;
  ANTECEDENT: BDI-proposition;
  CONSEQUENT: have;
  CONNECTION-SYMBOL: indicates;
  ATTRIBUTE:
    modal: {'Belief', 'Desire',
              'Intention', 'NOT'};
END RULE-TYPE ownership-rule;

RULE-TYPE unicity-rule;
  ANTECEDENT: BDI-proposition;
  CONSEQUENT: NOT have;
  CONNECTION-SYMBOL: indicates;
  ATTRIBUTE:
    modal: {'Belief', 'Desire',
              'Intention', 'NOT'};
END RULE-TYPE unicity-rule;

RULE-TYPE  benevolence-rule;
  ANTECEDENT: BDI-proposition;
  CONSEQUENT: Intention give;
  CONNECTION-SYMBOL: indicates;
  ATTRIBUTE:
    modal: {'Belief', 'Desire',
              'Intention', 'NOT'};
END RULE-TYPE   benevolence-rule;

RULE-TYPE parsimony-rule;
  ANTECEDENT: BDI-proposition
    AND Belief ability-rule;
  CONSEQUENT: BDI-proposition;
  CONNECTION-SYMBOL: indicates;
END RULE-TYPE parsimony-rule;

RULE-TYPE reduction-rule;
  ANTECEDENT: BDI-proposition AND
    Belief ability-rule AND
          NOT Belief ability-rule;
  CONSEQUENT: BDI-proposition;
  CONNECTION-SYMBOL: indicates;
END RULE-TYPE reduction-rule;

RULE-TYPE unique-choice-rule;
  ANTECEDENT: BDI-proposition AND
    Belief ability-rule AND
              Belief ability-rule;
  CONSEQUENT: BDI-proposition;
    XOR BDI-proposition;
  CONNECTION-SYMBOL: indicates;
END RULE-TYPE unique-choice-rule;

RULE-TYPE bridge-rule;
  ANTECEDENT: BDI-proposition;
  CONSEQUENT: BDI-proposition;
  CONNECTION-SYMBOL: indicates;
END RULE-TYPE bridge-rule;
```

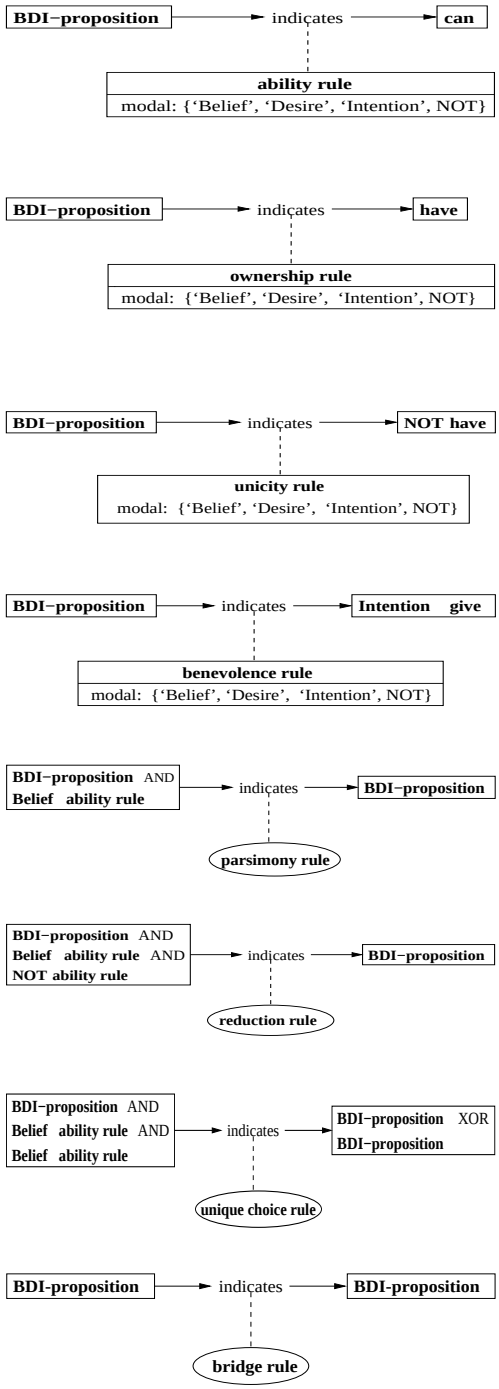


Figure 35. Rule type schema of the domain knowledge of persuader and persuadee agents.

describe an ability of a negotiating agent. Concept **have** is used to describe a resource or thing that a negotiating agent has. Concepts **give**, **ask** and **tell** are used to describe an action that a negotiating agent performs upon the another agent. Concept **BDI-proposition** is used to describe a BDI proposition.

The specification of rule types used by the persuader and persuadee agents is shown in Figure

```
KNOWLEDGE-BASE persuader-BDI-base;
  USES:
    can, have, BDI-proposition FROM concept-schema;
    ability-rule FROM ability-rule-schema;

INSTANCE can-hang-picture;
  INSTANCE OF: can;
  ATTRIBUTES:
    agent-name: 'persuader';
    can-do-thing: 'hang-picture';
END INSTANCE

INSTANCE have-picture;
  INSTANCE OF: have;
  ATTRIBUTES:
    owner-name: 'persuader';
    belongings: 'picture';
END INSTANCE

INSTANCE have-screw;
  INSTANCE OF: have;
  ATTRIBUTES:
    agent-name: 'persuader';
    belongings: 'screw';
END INSTANCE

INSTANCE have-hammer;
  INSTANCE OF: have;
  ATTRIBUTES:
    owner-name: 'persuader';
    belongings: 'hammer';
END INSTANCE

INSTANCE have-screw-driver;
  INSTANCE OF: have;
  ATTRIBUTES:
    owner-name: 'persuader';
    belongings: 'screw-driver';
END INSTANCE

INSTANCE have-nail;
  INSTANCE OF: have;
  ATTRIBUTES:
    owner-name: 'persuadee';
    belongings: 'nail';
END INSTANCE

INSTANCE wish;
  INSTANCE OF: BDI-proposition;
  ATTRIBUTES:
    proposition: can-hang-picture;
    prefix: Intention;
END INSTANCE

INSTANCE believe-having-picture;
  INSTANCE OF: BDI-proposition;
  ATTRIBUTES:
    proposition: have-picture;
    prefix: Belief
END INSTANCE

INSTANCE believe-having-screw;
  INSTANCE OF: BDI-proposition;
  ATTRIBUTES:
    proposition: have-screw;
    prefix: Belief
END INSTANCE

INSTANCE believe-having-hammer;
  INSTANCE OF: BDI-proposition;
  ATTRIBUTES:
    proposition: have-hammer;
    prefix: Belief
END INSTANCE

INSTANCE believe-having-screw-driver;
  INSTANCE OF: BDI-proposition;
  ATTRIBUTES:
    proposition: have-screw-driver;
    prefix: Belief
END INSTANCE

INSTANCE believe-having-nail;
  INSTANCE OF: BDI-proposition;
  ATTRIBUTES:
    proposition: have-nail;
    prefix: Belief
END INSTANCE

Belief(have.owner-name=X AND have.belongings='hammer' AND
       have.belongings='nail' AND have.belongings='picture'
         INDICATES
       can.agent-name=X AND can.can-do-thing='hang-picture')

Belief(have.owner-name=X AND have.belongings='screw-driver' AND
       have.belongings='screw' AND have.belongings='mirror'
         INDICATES
       can.agent-name=X AND can.can-do-thing='hang-mirror')

END KNOWLEDGE-BASE persuader-BDI-base;
```

FIGURE 36. Specification of persuader's BDI base.

35. There are four classes of rule types. The first class consists of only one rule type, **ability-rule**, used to describe what resources are needed if an agent is able to do something. For example, if an agent has a hammer, a nail and a picture then it can hang a picture. The second class consists of three rule types: **ownership-rule**, **unicity-rule** and **benevolence-rule**, which form a simple theory of action that integrates a model of the available resources with their planning capability mechanism. The planning mechanism is described by the third class of rule types **parsimony-rule**, **reduction-rule** and **unique-choice-rule**. In crude terms, such a rule means that when an agent believes that it has the intention of doing something and has a rule for achieving that intention, then the pre-conditions of the rule becomes a new intention. The final class consists of only one rule type **bridge-rule** that links inter-agent communication and the agent's internal state.

```
KNOWLEDGE-BASE persuadee-BDI-base;
  USES:
    can, have, BDI-proposition FROM concept-schema;
    ability-rule FROM  ability-rule-schema;

INSTANCE can-hang-mirror;
  INSTANCE OF: can;
  ATTRIBUTES:
    agent-name: 'persuadee';
    can-do-thing: 'hang-mirror';
END INSTANCE

INSTANCE have-mirror;
  INSTANCE OF: have;
  ATTRIBUTES:
    owner-name: 'persuadee';
    belongings: 'mirror';
END INSTANCE

INSTANCE have-nail;
  INSTANCE OF: have;
  ATTRIBUTES:
    owner-name: 'persuadee';
    belongings: 'nail';
END INSTANCE

INSTANCE wish;
  INSTANCE OF: BDI-proposition;
  ATTRIBUTES:
    proposition: can-hang-mirror;
    prefix: Intention;
END INSTANCE

INSTANCE believe-having-mirror;
  INSTANCE OF: BDI-proposition;
  ATTRIBUTES:
    proposition: have-mirror;
    prefix: Belief
END INSTANCE

INSTANCE believe-having-nail;
  INSTANCE OF: BDI-proposition;
  ATTRIBUTES:
    proposition: have-nail;
    prefix: Belief
END INSTANCE

Belief(have.owner=X AND have.belongings='hammer' AND
       have.belongings='nail' AND have.belongings='mirror'
         INDICATES
       can.agent-name=X AND can.can-do-thing='hang-mirror')

END KNOWLEDGE-BASE persuadee-BDI-base;
```

Figure 37. Specification of persuadee's BDI base.

```
KNOWLEDGE-BASE action-theory;
  USES:
    have, give, ask, BDI-proposition FROM concept-schema;
    ownership-rule, unicity-rule, benevolence-rule FROM action-schema;

Belief(have.owner-name=X AND have.belongings=Z AND
       give.giver-name=X AND give.receiver-name=Y AND give.thing=Z
         INDICATES
       have.owner-name=Y AND have.belongings=Z)

Belief(have.owner-name=X AND have.belongings=Z AND
       give.giver-name=X AND give.receiver-name=Y AND give.thing=Z
         INDICATES
       NOT(have.owner-name=X AND have.belongings=Z))

Belief(have.owner-name=Y AND have.belongings=Z AND
       NOT Intention(have.owner-name=Y AND have.belongings=Z) AND
       ask.asker-name=X AND ask.responder-name=Y
       AND ask.thing=(give.giver-name=Y AND give.receiver=X AND give.thing=Z)
         INDICATES
       Intention (give.giver-name=Y AND give.receiver=X AND give.thing=Z))

END KNOWLEDGE-BASE action-theory;
```

Figure 38. Specification of domain knowledge base for theory of action.

After giving the schema of domain concepts and rule types, we can give the specification of the domain knowledge bases now. In total, there are five knowledge bases. The persuader and persuadee have their own BDI bases and planning rules, but share knowledge bases **action-theory** and **bridge-rules**, both of which are domain independent.

The specification of the persuader's BDI base is shown in Figure 36. It means: the persuader has a picture, a screw, a hammer and a screwdriver; the persuader believes that the persuadee has a nail; the persuader knows that if a person has a hammer, a nail and a picture then the person can hang the picture and if a person has a screw, a screwdriver and a mirror then the person can hang the mirror; and the persuader intends to hang a picture.

```
KNOWLEDGE-BASE persuader-planning-rules;
  USES: have, give, ask, BDI-proposition FROM concept-schema;
        ability-rule FROM ability-rule-schema;
        parsimony-rules, reduction-rules FROM planning-rule-schema;

/* Parsimony type rules*/

Belief(Belief(NOT Intention(can.agent-name=X AND can.can-do-thing='hang-picture')) AND
       Belief(have.owner=X AND have.thing='hammer' AND
                              have.thing='nail' AND have.thing='picture'
                 INDICATES
              can.agent-name=X AND can.can-do-thing='hang-picture')
            INDICATES
       NOT Belief(Intention (have.owner=X AND have.thing='hammer')) AND
       NOT Belief(Intention (have.owner=X AND have.thing='nail')) AND
       NOT Belief(Intention (have.owner=X AND have.thing='picture')))

Belief(Belief(NOT Intention(can.agent-name=X AND can.can-do-thing='hang-mirror')) AND
       Belief(have.owner=X AND have.thing='screw-driver' AND
                              have.thing='mirror' AND have.thing='mirror'
                 INDICATES
              can.agent-name=X AND can.can-do-thing='hang-mirror')
            INDICATES
       NOT Belief(Intention (have.owner=X AND have.thing='screw-driver')) AND
       NOT Belief(Intention (have.owner=X AND have.thing='screw')) AND
       NOT Belief(Intention (have.owner=X AND have.thing='mirror')))

/* Reduction type rules */

Belief(Belief(Intention(can.agent-name=X AND can.can-do-thing='hang-picture')) AND
       Belief(have.owner=X AND have.thing='hammer' AND
                              have.thing='nail' AND have.thing='picture'
                 INDICATES
              can.agent-name=X AND can.can-do-thing='hang-picture')
            INDICATES
       Belief(Intention (have.owner=X AND have.thing='hammer')) AND
       Belief(Intention (have.owner=X AND have.thing='nail')) AND
       Belief(Intention (have.owner=X AND have.thing='picture')))

Belief(Belief(Intention(can.agent-name=X AND can.can-do-thing='hang-mirror')) AND
       Belief(have.owner=X AND have.thing='screw-driver' AND
                              have.thing='mirror' AND have.thing='mirror'
                  INDICATES
              can.agent-name=X AND can.can-do-thing='hang-mirror')
            INDICATES
       Belief(Intention (have.owner=X AND have.thing='screw-driver')) AND
       Belief(Intention (have.owner=X AND have.thing='screw')) AND
       Belief(Intention (have.owner=X AND have.thing='mirror')))

END KNOWLEDGE-BASE persuader-planing-rule;
```

FIGURE 39. Specification of the persuader's planning rules.

The specification of the persuadee's BDI base is shown in Figure 37. It means: the persuadee has a mirror and a nail; the persuadee knows that if a person has a hammer, a nail and a mirror then the person can hang the mirror; and the persuadee intends to hang a mirror.

The specification of knowledge base **action-theory** is shown in Figure 38. The first rule means that after X gives Z to Y, Y has Z. The second rule means that after X gives Z away, it no longer owns it. The third rule means that if an agent has something not in need for him then he likes to give the thing to another agent who asks him for the thing.

The specifications of the planning rules of the persuader and persuadee agents are shown in Figures 39 and 40, respectively. The parsimony type rules mean: if an agent believes that it does not intend something, it does not believe that it will intend the means to achieve it. The reduction type rules mean: if there is only one way to achieve an intention, an agent adopts the intention of achieving its preconditions.

The specifications of the bridge rules shared by the persuader and persuadee agents are shown in Figure 41. The first rule means that if an agent wants another agent to give it something then it asks the other for the thing. The second rule means that if an agent intends to give another agent something then it will tell the other the fact. The third rule means that if an agent is told something

```
KNOWLEDGE-BASE persuadee-planning-rules;
  USES: have, give, ask, BDI-proposition FROM concept-schema;
        ability-rule FROM ability-rule-schema;
        parsimony-rule, reduction-rule, unique-choice-rule FROM planning-rule-schema;

/* Parsimony type rules */

Belief(Belief(NOT Intention(can.agent-name=X AND can.can-do-thing='hang-mirror')) AND
       Belief(have.owner-name=X AND have.belongings='hammer' AND
                                    have.belongings='nail' AND have.belongings='mirror'
                  INDICATES
              can.agent-name=X AND can.can-do-thing='hang-mirror')
            INDICATES
       NOT Belief(Intention (have.owner-name=X AND have.belongings='hammer')) AND
       NOT Belief(Intention (have.owner-name=X AND have.belongings='nail')) AND
       NOT Belief(Intention (have.owner-name=X AND have.belongings='mirror')))

/* Reduction rules */

Belief(Belief(Intention(can.agent-name=X AND can.can-do-thing='hang-mirror')) AND
       Belief(have.owner-name=X AND have.belongings='hammer' AND
                                    have.belongings='nail' AND have.belongings='mirror'
                    INDICATES
                 can.agent-name=X AND can.can-do-thing='hang-mirror')
            INDICATES
       Belief(Intention (have.owner-name=X AND have.belongings='hammer')) AND
       Belief(Intention (have.owner-name=X AND have.belongings='nail')) AND
       Belief(Intention (have.owner-name=X AND have.belongings='mirror')))

END KNOWLEDGE-BASE persuadee-planing-rules;
```

Figure 40. Specification of the persuadee's planning rules.

```
KNOWLEDGE-BASE bridge-rules;
  USES: give, ask, tell, BDI-proposition FROM concept-schema;
        bridge-rule FROM bridge-rule-schema;

Intention(give.giver-name=X AND give.receiver-name=Y AND give.thing=Z)
  INDICATES
ask.asker-name=Y AND ask.responder-name=X AND
ask.thing=(give.giver-name=X AND give.receiver-name=Y AND give.thing=Z)

Intention(give.giver-name=X AND give.receiver-name=Y AND give.thing=Z)
  INDICATES
tell.teller-name=X AND tell.hearer-name=Y AND
tell.thing=(give.giver-name=X AND give.receiver-name=Y AND give.thing=Z)

tell.teller-name=X AND tell.hearer-name=Y AND
tell.thing=Belief(BDI-proposition)
  INDICATES
Belief(BDI-proposition)

Intention(BDI-proposition)
  INDICATES
Belief(Intention(BDI-proposition))

Intention(BDI-proposition)
  INDICATES
Belief(NOT Intention(BDI-proposition))

Belief(Intention(BDI-proposition))
  INDICATES
Intention(BDI-proposition)

END KNOWLEDGE-BASE bridge-rules;
```

Figure 41. Specification of the persuadee's bridge rules.

by another agent then it will believe that. The fourth and fifth rules mean that if an agent intends to do (or not do) something then it believes it has this intention. The sixth rule means that if an agent believes it has an intention then it adopts that intention.

6. RELATED WORK

Many researchers are working on Agent-Oriented Software Engineering (AOSE) (e.g. Iglesias *et al.*, 1999; Jennings, 2000; Zambonelli and Omicini, 2004; Georgeff, 2009). This is because, on the one hand, agents are being advocated as a next generation model for engineering open, complex, distributed systems (Wooldridge and Jennings, 1995; Jennings, 2001); on the other hand the task of agent system development is not easy especially without the help from software engineering environments (Lin *et al.*, 2007). Moreover, some of AOSE methodologies (e.g. Clarke, 2006) have been proven successful in industrial use in various application domains.

The main types of AOSE methodology are:

- The ones which extend/adapt existing techniques or methodologies of object-oriented software engineering. This can be done because agents can be thought of as more complex objects—active ones. The well-known example of this type is Gaia (Wooldridge *et al.*, 2000; Zambonelli *et al.*, 2003), which models a multi-agent system as a set of roles with permissions, responsibilities, and protocols. More specifically, it provides a methodology for analysing the roles and the interaction between roles and mapping them into agent, service and acquaintance models. Generally speaking, the processes for developing a multi-agent system include early requirement analysis, late requirement analysis, architectural design, detailed design and implementation. Gaia just covers the middle two. MESSAGE (Caire *et al.*, 2001) enhanced, based on UML and Gaia, design and implementation processes. There are some alternative AOSE methodologies covering different processes. For example, KAOS (Darimont *et al.*, 1998; Damas *et al.*, 2005) focuses on the first two, AUML (Bauer and Odell, 2005) and the Petri-net based framework (Xu and Shatz, 2003) on the detailed design, TROPOS (Bresciani *et al.*, 2004) on all but the last one, and MaSE (DeLoach *et al.*, 2001; DeLoach, 2001), a goal-oriented methodology (Shen *et al.*, 2006) and ASPECS (Cossentinoand *et al.*, 2010) tried to cover all the processes of agent modelling, design and construction.
- The ones which build on existing modelling techniques from knowledge engineering. The agent systems often need to use knowledge in autonomously interacting with the environment and other agents. In other words, they can be viewed as a kind of knowledge based systems and so knowledge engineering methodology can be extended for developing agent systems. Two well-known examples of the extensions are CoMoMAS (Glaser, 1996) and MAS-CommonKADS (Iglesias *et al.*, 1996).

In general, we can see that different AOSE methodologies have different focuses and characteristics. However, this diversity might confuse the developers of practical applications. Thus, some effort has been made to build up a unified framework that can capture commonalities of different AOSE methodologies. For example, the FAML meta-model (Beydoun *et al.*, 2009) is proposed for future standardization of an agent modelling language. Also, some specific aspects of AOSE methodologies are elaborated further. For example, the work of (Hu *et al.*, 2009) proposes, for the implementation process, the algorithms of transformation from platform-independent models to platform-specific models, and from platform-specific models to code. And the work (Sardinha *et al.*, 2006) proposes an integrated framework of the specification language and development process (from requirement analysis to implementation).

However, all of these existing methodologies are not specific enough for building up negotiating agents because they fail to capture specific aspects of such agents: flexible, autonomous, decision making behaviour, and the richness of interactions during the course of negotiation. In particular, although in the area of AOSE some interaction, organization and role pattern templates have been identified (Oluyomi *et al.*, 2006, 2008), all of them lack sufficient details that would specifically help in developing negotiating agents. Although the work (Chella *et al.*, 2010) identifies a number of pattern templates for specific agent systems—a robot, there no negotiation is involved in.

In contrast, this paper identifies many reusable pattern templates of communication, negotiation situation analysis and response decision making, and provides a detailed description for them using

the CommonKADS specification language. In other words, we build up a pattern library for the design of negotiating agents. Moreover, our generic model tells the developers how to assemble these pattern components together to form a concrete, complete negotiating agent. We identify these templates by analyzing existing negotiating agent systems (e.g. Parsons *et al.*, 1998; Kowalczyk and Bui, 2000; Barbuceanu and Lo, 2001; Faratin *et al.*, 2002; He and Jennings, 2002; Luo *et al.*, 2003a; He *et al.*, 2003a, 2006; Cheng *et al.*, 2006; Skylogiannis *et al.*, 2007) that have had a high impact in the area of automated negotiation. So, by using these templates, reuse can certainly increase the quality of the final systems developed. And also like all reusing in software engineering, these templates can reduce the development cost of new negotiating agents. In the work of (Bartolini *et al.*, 2005), a software framework for automated negotiation has been proposed. Nonetheless, it is not based on a throughout analysis of high impact systems, nor is it based on knowledge engineering. Thus it fails to identify a rich library of modular and reusable templates for key components of decision analysis and making during the course of negotiation. Also, since it is not based on knowledge engineering principles, it captures little of the knowledge features of negotiation situation analysis and response decision making, which plays an essential role in negotiation. In fact, the focus of (Bartolini *et al.*, 2005) is mainly on the interaction protocol design of the agents and no reusable and modular pattern templates of the decision making and analysis method design are identified.

Speaking more generally, the work in this paper is mainly about the architecture design of negotiating agents. Following this process is the detailed designing: negotiation strategy design or acquisition. This is also an important aspect of negotiating agent development but very complicated. Many papers focus on this issue. In particular, Luo *et al.* (2003b, 2004, 2006) investigated how to help users design the trade-off strategies for multi-issue negotiating agents. Vytelingum *et al.* (2005) proposed a framework for designing negotiation (bidding) strategies for trading agents in auctions. In (Ludwig, 2008), the issue of a concession strategy design for negotiating agent is studied based on the Thomas-Kilmann Conflict Mode Instrument (a commonly used psychological assessment tool) (Blake and Mouton, 1964; Kilmann and Thomas, 1977). In (Lin *et al.*, 2009), a toolbox is provided, which can help facilitate the negotiation strategy design according to the performance evaluation of the negotiating agent strategies. On the background of trading agent competition, Manistersky *et al.* (2008) discussed the issue of how users change their design of strategy according to negotiating agent performance in an open environment. Unlike our work in this paper, no architecture design issues of negotiation agents are involved in all of these papers. Actually, their focus is to build up the knowledge bases that are used in a negotiating agent whose architecture is pre-determined.

In addition, our methodology shares some common ideas with Component Based Software Engineering (CBSE) (Heineman and Councill, 2001): identify reusable components/patterns and then put the pieces together to build a specific system. However, in the area of CBSE, few researchers try to apply it for building autonomous software agents, especially negotiating agents. And negotiating agents are knowledge intensive systems but in CBSE there are no facilities to handle the aspect of knowledge modelling and specification. That is why we need to base our methodology on knowledge engineering methodology.

7. CONCLUSIONS AND FUTURE WORK

Automated negotiation can be applied in any domain where multiple stakeholders with conflicting positions and even conflicting interests have to work together to get a job done. However, it is not easy to develop such systems without the aid of a software engineering methodology. Unfortunately, so far such a methodology has been missing. To remove the limitation, this paper develops a template based knowledge engineering methodology for building automated negotiation systems. It consists primarily of a generic knowledge model of a main task and a number of standardised templates which are modular and reusable components of the main task model. The different combination of templates (including their variants) can constitute different automated negotiation models. Actually, these templates function as blocks for building such systems, and the generic model functions as the blue print for assembling these blocks together to produce different automated negotiation models. Thus, with the help of the proposed methodology, it now becomes much easier to develop automated negotiation systems. Moreover, since these standardised components are identified from some high impact, typical negotiation systems which are selected from a wide variety of such systems, and the

generic model that captures overall structure and organisation of a negotiating agent is based on the analysis on many such systems, a negotiation system developed according to the methodology should be effective. As a result, the methodology has the potential to facilitate the widespread development of applications of automated negotiation.

Some limitations of this work could be addressed in the further work:

- Although most detailed settings of the methodology are primarily based on our analysis of bilateral negotiation systems, we believe the application scope of the whole methodology can be readily be extended to multilateral negotiation. However, this needs to be checked out further in the future work.
- Although the validity of our methodology is shown by an argumentation model (Parsons *et al.*, 1998) that has a high impact in the area of automated negotiation, we believe that our methodology is valid not only for the existing systems but also for new ones, especially complex business-to-business negotiations. Of course, in the future more work (especially running examples) is required to confirm this.
- From the viewpoint of software engineering, our methodology can be extended further. First, the requirement analysis and elicitation process should be included. Like (Damas *et al.*, 2005), existing technologies of knowledge acquisition (Milton, 2007) and machine learning (Alpaydin, 2004) could be employed for the requirement analysis process. Second, it would be helpful if the methodology could help the developers to find the proper components from the library of standardised templates according to the outcome of requirement analysis for specific applications. Third, the template library needs to be extended further to include more templates. Finally, based on the extended methodology, a software engineering environment, that supports the processes from requirement analysis to system coding, could be made available. In particular, in such an environment, there should be a tool to assist the users in specifying complex negotiation systems since as we shown in the argumentation example of Parsons *et al.* (1998) it could be very time consuming to apply the methodology to generate all the required specifications (although our methodology certainly make the whole thing easier).
- In this paper we identified what types of domain knowledge are usually used in manual negotiation approaches and automated negotiation models. On the basis, we can further develop techniques to acquire these types of domain knowledge from human users. This is important because in order that automated negotiation systems can negotiate faithfully on the behalf of their human users, the relevant domain knowledge must be properly acquired from the human users (Preist, 2001; Luo *et al.*, 2003b, 2004, 2006; Ludwig, 2008).
- Although the verification problem in developing complex software systems is as important as preparing the specification of a software system, it is beyond the scope of this paper. In the future, we will address this issue. In particular, in the area of knowledge engineering, Cornelissen *et al.* (1997) extended the idea of component reusing to the reuse of proofs for properties of components reused. Actually, it is a case study for diagnostic reasoning, but maybe its idea can be extended and applied, in our methodology, to investigate the properties of negotiating agents from the properties of its components reused. Of course, we could also extend some ideas from the area of software engineering, for example that of model-driven software verification (Holzmann and Joshi, 2004).

Acknowledgments

The authors would like to thank the anonymous referees for their comments which significantly improved the quality of the paper.

REFERENCES

Ai, M. (2002). *Handbook of Negotiation and Contract Signing*. Internal Mongolia Culture Publishing House. In Chinese.

Alpaydin, E. (2004). *Introduction to Machine Learning*. The MIT Press.

Bagüés, S., Mitic, J., Zeidler, A., Tejada, M., Matias, I., and Valdivielso, C. (2008). Obligations: Building a bridge between personal and enterprise privacy in pervasive computing. In S. Furnell, S. Katsikas, and A. Lioy, editors, *Trust, Privacy and Security in Digital Business*, volume 5185 of *Lecture Notes in Computer Science*, pages 173–184. Springer.

Barbuceanu, M. and Lo, W.-K. (2001). Multi-attribute utility theoretic negotiation for electronic commerce. In F. Dignum and U. Cortés, editors, *Agent-Mediated Electronic Commerce III*, volume 2003 of *Lecture Notes in Artificial Intelligence*, pages 15–30. Springer.

Bartolini, C., Preist, C., and Jennings, N. R. (2005). A software framework for automated negotiation. In *Software Engineering for Multi-Agent Systems III*, volume 3390 of *Lecture Notes in Computer Science*, pages 213–235. Springer.

Bauer, B. and Odell, J. (2005). UML 2.0 and agents: How to build agent-based systems with the new UML standard. *Engineering Applications of Artificial Intelligence*, **18**(2), 141–157.

Benjamins, V. (1993). *Problem Solving Methods for Diagnosis*. Ph.D. thesis, University of Amsterdam, Amsterdam.

Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J., Pavon, J., and Gonzalez-Perez, C. (2009). Faml: A generic metamodel for mas development. *IEEE Transactions on Software Engineering*, **35**(6), 841–863.

Blake, R. and Mouton, J. (1964). *The managerial Grid*. Gulf Publications, Houston.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., and Perini, A. (2004). Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, **8**(3), 203–336.

Caire, G., Leal, F., Chainho, P., Evans, R., Garijo, F., Gomez, J., Pavon, J., Kearney, P., Stark, J., and Massonet, P. (2001). Agent oriented analysis using MESSAGE/UML. In *Proceedings of the Second International Workshop on Agent-Oriented Software Engineering*, pages 101–108.

Cappiello, C., Comuzzi, M., and Plebani, P. (2007). On automated generation of web service level agreements. In J. Krogstie, A. Opdahl, and G. Sindre, editors, *Advanced Information Systems Engineering*, volume 4495 of *Lecture Notes in Computer Science*, pages 264–278. Springer.

Chao, K.-M., Younas, M., Godwin, N., and Sun, P.-C. (2006). Using automated negotiation for grid services. *International Journal of Wireless Information Networks*, **13**(2), 141–150.

Chella, A., Cossentino, M., Gaglio, S., Sabatucci, L., and Seidita, V. (2010). Agent-oriented software patterns for rapid and affordable robot programming. *Journal of Systems and Software*, **83**(4), 557–573.

Cheng, C.-B., Chan, C.-C., and Lin, K.-C. (2006). Intelligent agents for e-marketplace: Negotiation with issue trade-offs by fuzzy inference systems. *Decision Support Systems*, **42**(2), 626–638.

Clarke, D. (2006). Commercial experience with agent-oriented software engineering. In *Proceeding of 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 730–736.

Collin, P. (2001). *Dictionary of Business (Third Edition)*. Peter Collin Publishing.

Cornelissen, F., C.M.Jonker, and J.Treur (1997). *Compositional verification of knowledge-based systems: A case study for diagnostic reasoning*, volume 1319 of *Lecture Notes in Computer Science*, pages 65–80. Springer.

Cossentinoand, M., Gaud, N., Hilaire, V., Galland, S., and Koukam, A. (2010). Aspecs: An agent-oriented software process for engineering complex systems. *Autonomous Agents and Multi-Agent Systems*, **20**(2), 260–304.

Damas, C., Lambeau, B., Dupont, P., and van Lamsweerde, A. (2005). Generating annotated behavior models from end-user scenarios. *IEEE Transactions on Software Engineering*, **31**(12), 1056–1073.

Darimont, R., Delor, E., Massonet, P., and van Lamsweerde, A. (1998). GRAIL/KAOS: An environment for goal-driven requirements engineering. In *Proceedings of the 20th International Conference on Software Engineering*, volume 2, pages 58–62.

DeLoach, S. (2001). Analysis and design using MaSE and agentTool. In *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference*.

DeLoach, S., Wood, M., and Sparkman, C. (2001). Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, **11**(3), 231–258.

Egels-Zandèn, N. (2009). Tnc motives for signing international framework agreements: A continuous

bargaining model of stakeholder pressure. *Journal of Business Ethics*, **84**(4), 529–547.

Faratin, P., Sierra, C., and Jennings, N. R. (2002). Using similarity criteria to make issue tradeoffs in automated negotiations. *Artificial Intelligence*, **142**(2), 205–237.

Finin, T. and Labrou, Y. (1997). KQML as an agent communication language. In J. Bradshaw, editor, *Software Agents*, pages 291–316. AAAI Press and MIT Press.

Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). KQML as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management*, pages 456–463.

Fisher, R., Ury, W., and Patton, B. (1991). *Getting to yes: Negotiating an agreement without giving in*. Penguin Books. This is the revised 2nd edition. The first edition, unrevised, is published by Houghton Mifflin, 1981.

Georgeff, M. (2009). The gap between software engineering and multi-agent systems: Bridging the divide. *International Journal of Agent-Oriented Software Engineering*, **3**(4), 391–396.

Glaser, N. (1996). *Contribution to Knowledge Modelling in a Multi-Agent Framework: The CoMo-MAS Approach*. Ph.D. thesis, LUniverstitè Henri Poincarè, Nancy I, France.

Guan, S., Dong, X., Mei, Y., Wu, W., and Xue, Z. (2008). Towards automated trust negotiation for grids. In *Proceedings of the 2008 IEEE International Conference on Networking, Sensing and Control*, pages 154–159.

He, M. and Jennings, N. R. (2002). SouthamptonTAC: Designing a successful trading agent. In *Proceedings of the Fifth European Conference of Artificial Intelligence*, pages 8–11.

He, M., Leung, H. F., and Jennings, N. R. (2003a). A fuzzy logic based bidding strategy in continuous double auctions. *IEEE Transactions on Knowledge and Data Engineering*, **15**(6).

He, M., Jennings, N. R., and Leung, H. F. (2003b). On agent-mediated electronic commerce. *IEEE Transactions on Knowledge and Data Engineering*, **15**(4), 985–1003.

He, M., Rogers, A., Luo, X., and Jennings, N. R. (2006). Designing a successful trading agent for supply chain management. In *Proceedings of the Fifth International Conference on Autonomous Agents and Multi-Agent Systems*, pages 61–62, Hakodate, Japan.

Heineman, G. T. and Councill, W. T. (2001). *Component-based software engineering: Putting the pieces together*. ACM Press.

Hoffman, E., McCabe, K., and Shachat, K. (1994). Preferences, property rights, and anonymity in bargaining games. *Games and Economic Behavior*, **7**, 346–380.

Holzmann, G. J. and Joshi, R. (2004). *Model-Driven Software Verification*, volume 2989 of *Lecture Notes in Computer Science*, pages 76–91. Springer.

Hu, C., Mao, X., and Ning, H. (2009). Integrating model transformation in agent-oriented software engineering. In *Proceedings of 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, pages 62–65.

Huhns, M. N. and Stephens, L. M. (1999). Multiagent systems and societies of agents. In G. Weiss, editor, *Multiagent Systems*, pages 79–120. The MIT Press.

Iglesias, C. A., Garijo, M., Gonzàlez, J. C., and Velasco, J. R. (1996). A methodology proposal for mutiagent systems development extending CommonKADS. In *Proceedings of the Tenth KAW*, pages 345–360.

Iglesias, C. A., Garijo, M., and Gonzalez, J. C. (1999). A survey of agent-oriented methodologies. In J. P. Müller *et al.*, editors, *Intelligent Agents V: Agent Theories, Architectures, and Languages*, volume 1555 of *Lecture Notes in Computer Science*, pages 317–330. Springer.

Jennings, N. R. (2000). On agent-based software engineering. *Artificial Intelligence*, **117**(2), 277–296.

Jennings, N. R. (2001). An agent-based approach for building complex software systems. *Comms. of the ACM*, **44**(4), 35–41.

Kahneman, D. and Tversky, A. (1979). Prospect theory: An analysis of decision under risk. *Econometrica*, **47**, 263–290.

Keeney, R. and Raiffa, H. (1976). *Decision with Multiple Objectives: Preferences and Value Tradeoffs*. John Wiley & Sons.

Kersten, G. and Lai, H. (2007). Negotiation support and e-negotiation systems: An overview. *Group Decision and Negotiation*, **16**(6), 553–586.

Kilmann, R. and Thomas, K. (1977). Developing a forced-choice measure of conflict-handling: The mode instrument. *Educational and Psychological Measurement*, **37**, 309–325.

Koulouris, T., Spanoudakis, G., and Tsigkritis, T. (2007). Towards a framework for dynamic verification of peer-to-peer systems. In *Proceedings of the Second International Conference on Internet and Web Applications and Services*, pages 2–12.

Koumoutsos, G. and Thramboulidis, K. (2009). Towards a knowledge-base framework for complex, proactive and service-oriented e-negotiation systems. *Electronic Commerce Research*, **9**(4), 317–349.

Kowalczyk, R. and Bui, V. (2000). On constraint-based reasoning in e-negotiation agents. In F. Dignum and U. Cortés, editors, *Agent-Mediated E-Commerce III*, volume 2003 of *Lecture Notes in Artificial Intelligence*, pages 31–46. Springer.

Lee, K.-C. and Kwon, S.-J. (2006). The use of cognitive maps and case-based reasoning for B2B. *Journal of Management Information Systems*, **22**(4), 337–376.

Lin, C.-E., Kavi, K., Sheldon, F., Deley, K., and Abercrombie, R. (2007). A methodology to evaluate agent oriented software engineering techniques. In *Proceedings of the 40th Hawaii International Conference on System Sciences*, page 60a.

Lin, R., Kraus, S., Tykhonov, D., Hendriks, K., and Jonker, C. (2009). Supporting the design of general automated negotiators. In *Proceedings of the Second International Workshop on Agent-based Complex Automated Negotiations*, Budapest, Hungary.

Lomuscio, A. R., Wooldridge, M., and Jennings, N. R. (2003). A classification scheme for negotiation in electronic commerce. *International Journal of Decision and Negotiation*, **12**(1), 31–56.

Loutaa, M., Roussakib, I., and Pechlivanos, L. (2008). An intelligent agent negotiation strategy in the electronic marketplace environment. *European Journal of Operational Research*, **187**(3), 1327–1345.

Ludwig, S. (2008). Agent-based assistant for e-negotiations. In A. An, S. Matwin, Z. Raś, and D. Ślezak, editors, *Foundations of Intelligent Systems*, volume 4994 of *Lecture Notes in Artificial Intelligence*, pages 514–524. Springer.

Luo, X. and Jennings, N. R. (2007). A spectrum of compromise aggregation operators for multi-attribute decision making. *Artificial Intelligence*, **171**(2-3), 161–184.

Luo, X., Jennings, N. R., Shadbolt, N., Leung, H. F., and Lee, J. H. M. (2003a). A fuzzy constraint based model for bilateral, multi-issue negotiation in semi-competitive environments. *Artificial Intelligence*, **148**(1-2), 53–102.

Luo, X., Jennings, N. R., and Shadbolt, N. (2003b). Knowledge-based acquisition of tradeoff preferences for negotiating agents. In *Proceedings of the 5th International Conference on Electronic Commerce*, pages 138–144, Pittsburgh, USA.

Luo, X., Jennings, N. R., and Shadbolt, N. (2004). Acquiring tradeoff preferences for automated negotiations: A case study. In *Agent-Mediated Electronic Commerce V: Designing Mechanisms and Systems*, volume 3048 of *Lecture Notes in Computer Science*, pages 37–55. Springer.

Luo, X., Jennings, N. R., and Shadbolt, N. (2006). Acquiring user tradeoff strategies and preferences for negotiating agents: A default-then-adjust method. *International Journal of Human Computer Studies*, **64**(4), 304–321.

Manistersky, E., Lin, R., and Kraus, S. (2008). Understanding how people design trading agents over time. In *Proceedings of the Seventh International Conference on Automous Agents and Multiagent systems*, pages 1593–1596.

McCauley-Bell, P. (1999). Intelligent agent characterization and uncertainty management with fuzzy set theory: A tool to support early supplier integration. *Journal of Intelligent Manufacturing*, **10**, 135–147.

Milton, N. (2007). *Knowledge Acquisition in Practice: A Step-by-step Guide*. Springer.

Murch, R. and Johson, T. (1999). *Intelligent Software Agents*. Prentice-Hall.

Nash, J. (1950). The bargaining problem. *Econometrica*, **18**(2), 155–162.

Neumann, J. V. and Morgenstern, O. (1944). *Theory of Games and Economic Behaviour*. Princeton University Press.

O'Brien, P. and Nicol, R. (1998). FIPA: Towards a standard for software agents. *BT Technology Journal*, **16**(3), 51–59.

Oluyomi, A., Karunasekera, S., and Sterling, L. (2006). Design of agent-oriented pattern templates. In *Proceedings of Australian Software Engineering Conference*, pages 113–121.

Oluyomi, A., Karunasekera, S., and Sterling, L. (2008). Description templates for agent-oriented

patterns. *Journal of Systems and Software*, **81**(1), 20–36.
Park, S. and Yang, S.-B. (2008). An efficient multilateral negotiation system for pervasive computing environments. *Engineering Applications of Artificial Intelligence*, **21**(8), 633–643.
Parsons, S., Sierra, C., and Jennings, N. R. (1998). Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, **8**(3), 261–292.
Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.
Peasall, J. (2001). *The New Oxford English Dictionary*. Oxford University Press.
Preist, C. (2001). Agent mediated electronic commerce at HP labs, Bristol. *Agentlink News*, (7).
Pruitt, D. (1981). *Negotiation Behavior*. Academic Press.
Ragone, A., Noia, T., Sciascio, E., and Donini, F. (2008). Logic-based automated multi-issue bilateral negotiation in peer-to-peer e-marketplaces. *Autonomous Agents and Multi-Agent Systems*, **16**(3), 249–270.
Rahwan, I., Ramchurn, S. D., Jennings, N. R., McBurney, P., Parsons, S., and Sonenberg, L. (2004). Argumentation-based negotiation. *The Knowledge Engineering Review*, **18**(4), 343–375.
Raiffa, H. (1982). *The Art and Science of Negotiation*. Harvard University Press, Cambridge, USA. Sixteenth printing, 2002.
Rubinstein, A. (1982). Perfect equilibrium in a bargaining model. *Econometrica*, **50**(1), 97–109.
Sardinha, J., Choren, R., da Silva, V., Milidiù, R., and de Lucena, C. (2006). A combined specification language and development framework for agent-based application engineering. *Journal of Systems and Software*, **79**(11), 1565–1577.
Schreiber, G., Akkermans, H., Anjewierden, A., Hoog, R., N. Shadbolt, W. V., and Wielinga, B. (1999). Chapter 5: Knowledge model construction. In *Knowledge Engineering and Management: The CommonKADS Methodology*, pages 167–186. The MIT Press.
Schreiber, G., Akkermans, H., Anjewierden, A., Hoog, R., Shadbolt, N., Velde, W., and Wielinga, B. (2000). *Knowledge Engineering and Management: The CommonKADS Methodology*. The MIT Press.
Searle, J. (1969). *Speech Acts*. Cambridge University Press.
Shen, Z., Miao, C., Gay, R., and Li, D. (2006). Goal-oriented methodology for agent system development. *IEICE TRANSACTIONS on Information and Systems*, **E89-D**(4), 1413–1420.
Skylogiannis, T., Antoniou, G., Bassiliades, N., Governatori, G., and Bikakis, A. (2007). DR-NEGOTIATE — A system for automated agent negotiation with defeasible logic-based strategies. *Data & Knowledge Engineering*, **63**(2), 362–380.
Unt, I. (1999). *Negotiation Without A Loser*. Copenhagen Business School.
Vytelingum, P., Dash, R. K., He, M., and Jennings, N. R. (2005). A framework for designing strategies for trading agents. In *Agent Mediated Electronic Commerce VII*, volume 3937 of *Lecture Notes in Artificial Intelligence*, pages 171–186. Springer.
Wang, M., Wang, H., Vogel, D., Kumar, K., and Chiu, D. (2009). Agent-based negotiation and decision making for dynamic supply chain formation. *Engineering Applications of Artificial Intelligence*, **22**(7), 1046–1055.
Wellman, M., Greenwald, A., and Stone, P. (2007). *Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition*. MIT Press, Cambridge MA, USA.
Wong, W., Zhang, D., and Kara-Ali, M. (2000a). Negotiating with experience. In *Proceedings of AAAI2000 Workshop on Knowledge-Based Electronic Markets*, pages 85–90.
Wong, W., Zhang, D., and Kara-Ali, M. (2000b). Towards an experience-based negotiation agent. In *Proceedings of the Fourth International Workshop on Cooperative Information Agents*, Boston.
Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, **10**(2), 115–152.
Wooldridge, M. and Parsons, S. (2000). Languages for negotiation. In *Proceedings of the Fourteenth European Conference on Artificial Intelligence*, Berlin, Humboldt University, Germany.
Wooldridge, M., Jennings, N. R., and Kinny, D. (2000). The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, **3**(3), 285–312.
Xu, H. and Shatz, S. M. (2003). A framework for model-based design of agent-oriented software. *IEEE Transactions on Software Engineering*, **29**(1), 15–30.
Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, **8**, 338–353.

Zadeh, L. A. (1975). The calculus of fuzzy restrictions. In L. A. Zadeh *et al.*, editors, *Fuzzy Sets and Applications to Cognitive and Decision Making Processes*, pages 1–39. Academic Press.

Zambonelli, F. and Omicini, A. (2004). Challenges and research directions in agent-oriented software engineering. *Autonomous Agents and Multi-Agent Systems*, **9**(3), 253–283.

Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2003). Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology*, **12**(3), 317–370.

Zeng, D. and Sycara, K. (1997). How can an agent learn to negotiate? In J. P. Müller, M. J. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III: Agent Theories, Architectures, and Languages*, volume 1193 of *Lecture Notes in Artificial Intelligence*, pages 233–244. Springer.