Goal-Oriented Interaction Protocols

Lars Braubach and Alexander Pokahr

Distributed Systems and Information Systems
Computer Science Department, University of Hamburg
{braubach | pokahr}@informatik.uni-hamburg.de

Abstract. Developing agent applications is a complex and difficult task due to a variety of reasons. One key aspect making multi-agent systems more complicated than traditional applications is that interaction behavior is based on elaborate communication forms such as negotiations instead of simple method calls. Aimed at facilitating the specification and usage of agent communication, agent research resulted e.g. in the definition and standardization of several general purpose interaction protocols such as contract-net or English auction. Nevertheless, the usage of these valuable interaction patterns currently forces developers to concentrate on the details of message passing instead of thinking in terms of the application domain. To alleviate this problem in this paper a goal-oriented approach is proposed, which hides message passing details allowing developers to concentrate on the domain aspects of protocols. The new approach is based on the BDI agent model and is implemented within the Jadex agent framework. The advantages of the goal-based interaction handling are further illustrated by an example application.

1 Introduction

The ability to interact with each other is generally accepted as one of the important properties of software agents [20]. Interaction is required as a means to coordinate the actions of the individual agents of a multi-agent system (MAS) in order to achieve overall system goals and to improve the effectiveness of the system [11]. Despite the importance of interacting agents, realizing the necessary interactions is one main source of difficulties during the development of a multi-agent system. These difficulties stem from the fact that, unlike traditional systems, multi-agent systems are usually inherently distributed and asynchroneous without any central control. Regarding the design and implementation of interactions in a multi-agent system, developers are therefore confronted with a multitude of conceptual and implementation related questions such as:

Macro Level	1. What are the objectives behind the interaction? 2. What are the characteristic properties of the interaction? 3. How can the interaction be described and analyzed?
Micro Level	4. What are the objectives of the interacting agents? 5. How is the interaction related to the agent architecture? 6. How is the interaction related to domain-specific behavior?

According to Ferber [11], interaction can be viewed from a macro level perspective (i.e. for the MAS as a whole) as well as on the micro level (i.e. considering the individual agents). In the macro level perspective, the objectives of the system as a whole (question 1) need to be considered that aim at coordinating the behaviour of individual agents towards establishing some global properties of the system (e.g. using market-based coordination mechanisms to achieve fair pricing of traded goods). Properties of interactions (question 2) can be classified according to different criteria related to the dialog structure, which can be defined in advance as a fixed sequence of messages with only a limited number of alternatives (called interaction protocols) or evolve dynamically according to loose regulations allowing flexible reactions of the participants. To support a systematic construction of interactions adequate macro level description means are necessary (question 3) that contain information about the interaction objective as well as the specific properties. Restricting the topic to interaction protocols a variety of techniques has been proposed in the field of multi-agent systems, such as the well known AUML sequence diagrams [1]. Besides the description also the analysis is important for the validation of the system, where approaches range from formal verification to runtime monitoring of agent behavior.

The micro level perspective deals with questions regarding the implementation of the individual agents. To implement the local decision processes of the individual agents, the developer has to lay down the individual objectives (question 4) that apply to the steps of the interaction. After deciding on the agents objectives, the developer is confronted with numerous implementation choices. Nowadays, there exists a vast number of more or less mature software frameworks supporting developers in building complex multi-agent systems [3]. These frameworks employ different agent architectures used to define the behavior of the agents, which are commonly based on abstract mentalistic notions (e.g. the BDI model [4]) or on simple task-centric concepts derived from software engineering needs. Therefore, the question arises, how these internal agent architectures relate to separately designed interactions (question 5). Finally, the developer has to solve the problem of how to integrate the domain-specific application logic with the previously designed interaction flow (question 6).

Despite the importance of all these questions and also of the link between both levels this paper focuses on micro level questions (4-6) and is organized as follows: In section 2, related work regarding the support for interactions in multi-agent systems is presented. Section 3 describes a new approach to the implementation of protocols for BDI-style agents, employing goals as a central concept for establishing the connection between (external) interactions and (internal) reasoning. In section 4, the realization of this approach within the Jadex agent framework is explained and demonstrated with an illustrative example in section 5. The paper concludes with a summary and an outlook in section 6.

2 Related Work

Research that aims at improving the agent interaction realization can be coarsely divided in *protocol-based interactions* and *flexible interactions*, whereby *protocol-*

based interactions can be subdivided in *generator-* and *interpreter-oriented approaches*. Generally, *generator approaches* allow transforming protocol descriptions into executable code specifications. E.g. in [9] a tool is presented for the automatic transition of AUML protocol descriptions into JADE behaviours. Further work supports other agent platforms such as Mulan or AgentFactory as well [7,16]. Most generator approaches produce initial code skeletons and leave the connection with the domain logic to the developer (question 6), leading to code maintainance problems as generic protocol code and domain-specific code are highly intertwined. As those approaches are mostly targeted towards simple task-centric agent platforms, the connection of protocols with the target agent architecture is currently also neglected (question 5). The alternative are *interpreter-oriented approaches* that process protocol descriptions at runtime, therefore requiring a generic mechanism that allows integrating protocol execution with domain-specific behaviour (question 6). E.g. in [10] and [17] interpreters based on (different) formalizations of AUML are proposed, whereby the domain-dependent parts are connected via method invocations whenever a message is received or has to be sent. These approaches still focus on message sequences and do not provide domain level abstractions. In addition, interpreter approaches are usually architecture-independent and therefore do not exploit the full potential of a specific agent architecture (question 5).

Flexibility of interactions is achieved by relaxing the constraints that exist in using predefined protocols, leading to more fault-tolerant and hence robust communications, which are driven by the interests of the communication participants and not by predefined sequences of message patterns. E.g. in Hermes [8] a goal-oriented approach is proposed that focuses on the macro level questions (1-3) and aims at decomposing interactions into a hierarchy of interaction goals, where each leaf goal represents a partial interaction. These goals enable failure recovery and rollbacks in case of unexpected communication outcomes. Other approaches exploit the message semantics, i.e. performative and content, to determine how to react to a message (see e.g. the JADE semantic agent [2] and the LS/TS SemCom [19] architecture). These approaches target question 5 and to some extend also 4 and 6, but usually employ custom control structures instead of established agent architectures such as BDI. As standardized interaction protocols have proven their value in design and implementation of agent interactions, flexible interaction approaches should be regarded as augmentation and not as a replacement for protocol-based interaction. While Hermes does not focus on how the partial interactions represented by leaf interaction goals should be implemented, the sematic communication approaches currently do not allow the use of predefined interaction protocols.

An ideal approach should address all the questions posed in the introduction. Specifically, we think that a unfied perspective is required that considers protocols with respect to their objectives, agent architecture and domain connection. Only such a holistic view, that is achieved by none of the presented research efforts, will enable an abstract domain-centered perception of interaction protocols. The approach presented in this paper can be regarded as one step in this direction and is motivated by the interpreter-based perspective.

3 Goal-Oriented Protocols Approach

Interaction protocols have gained high attention in the context of multi-agent communication, as they capture established best-practices that facilitate the realization of interaction-based application scenarios. Standardized protocols concretize abstract mechanisms specifically designed for generic domain-independent use-cases. Mechanism-dependent properties help deciding which protocols to use in a concrete project setting. E.g. Wooldridge [20] proposes several general criteria such as social welfare, guaranteed success or individual rationality that can be used for comparing candidate mechanisms. When implementing the corresponding protocols, developers should be enabled to concentrate on the domain-aspects of protocols abstracting away from their concrete realization via message passing. In the following it will be discussed how the micro level questions can be used to deduce further requirements on protocol support.

Agent objectives (question 4) that have been settled during design should be conserved within the implementation, providing an intentional stance [14] with respect to the conversational behavior of agents, which facilitates explainability and predictability of communication. Moreover, agent developers should be enabled to use the same concepts offered by the agent architecture also for the implementation of agent conversations (question 5). A seamless integration allows exploiting the full potential of the architecture and requires the protocol support to be specifically taylored towards a suitable target architecture. Finally, the integration of domain logic with the generic protocols (question 6) should allow a clear separation of both aspects, facilitating the independent further development of both aspects and e.g. understandability and maintainability of application code. The integration should be done on an abstract level promoting the domain-view and hiding message level details.

Starting point for the approach presented in this paper is the belief-desire-intention model of agency (BDI-model) [4]. Interaction goals are introduced for expressing the objectives that the individual communication partners exhibit, serving as the connectives between a generic interaction protocol and the BDI architecture. Goals are advantageous, because they represent the motivations of an agent in an abstract manner, intentionally leaving open the means that could be used for their pursuit. This allows to capture the abstract objectives that control the agents participation during an ongoing conversation without considering concrete activities or low-level message handling. Details regarding the deduction of interaction goals from protocol descriptions and the integration of domain specific behavior by the application developer are presented next.

3.1 Domain Interaction Analysis

In this section, a process is proposed allowing to deduce descriptions of goal-oriented interaction protocols by analyzing normal AUML protocol representations [1]. In Fig. 1 a schematic view of an AUML-based goal-oriented interaction protocol is depicted. In contrast to the original AUML representation each role is divided into two distinct parts. The original protocol layer (middle) is domain-independent and responsible solely for dialog control and execution of protocol

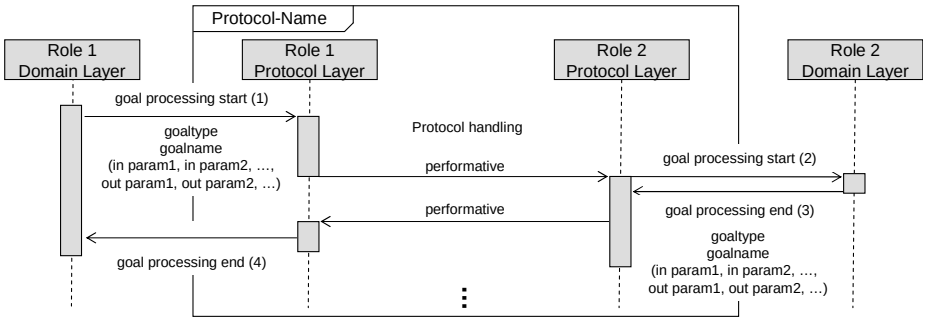

Fig. 1. Protocol analysis with AUML interaction diagrams

specific actions. The protocol layer is augmented by the newly introduced domain layer (left resp. right) which encapsulates domain-relevant actions. The separation of protocols into these distinct parts helps to make explicit the interfaces between the domain and protocol parts. The first task now consists in finding out at which positions in each role of the protocol domain-specific activities are necessary, who initiates these activities and when they will be finished. In the diagram for each such action a goal description needs to be defined. This description consists of a pair of arrows indicating the beginning and ending of the domain activity and which part of a role initiates the activity within the other part. E.g. in the schematic view one can see that the domain layer of role 1 initiates the protocol execution via activity activation within the protocol layer of role 1 and finally fetches the results of its execution (arrows 1 and 4). Similarly, during protocol execution the protocol layer of role 2 needs a domain activity being executed and delegates it to the domain layer of role 2 (arrows 2 and 3). In a second step the goal descriptions need to be refined by specifying the more concrete goal signatures. This means it has to be analyzed what kind of domain activity is needed and which information needs to be transferred forth and back between the domain and protocol part of a role. The kind of activity determines the general goal type to be used, e.g. if information needs to be retrieved a *query* goal would be appropriate and if a task needs to be executed an *achieve* goal would be a good choice. An overview of the most commonly used goal types and their application can be found in [6] and is supported by influential methodologies and modelling approaches such as KAOS [18] and Tropos [13]. In the last step the signatures will be completed by adding detailed in- and out-parameter descriptions which have to be deduced from the informal activity descriptions.

3.2 Integration of Domain Behavior

The domain interaction analysis process results in the specification of the individual interaction goals of the participating agents. Interaction goals solve the problem of connecting protocol execution and agent architecture (question 5) as the agent applies its general reasoning strategies to handle these goals. From the viewpoint of the protocol execution, these goals are abstract, i.e., the behavior triggered by these goals is transparent. For the application developer, the in-

teraction goals represent the access point for supplying the domain-dependent behavior (question 6), capturing the activities to be performed for each interaction goal. Goal parameters provide access to the relevant domain and communication data (e.g. the subject-matter of a negotiation), which can be used while executing arbitrary domain tasks. After finishing the domain tasks, the results are made available in the out-parameters of a goal. The goal specification therefore provides a clean interface, allowing the domain behavior accessing necessary information and making results of domain tasks available.

4 Realization within Jadex

The goal-oriented protocols approach is realized within the Jadex BDI agent system [15]. Jadex aims at facilitating the development of multi-agent systems by introducing abstract notions such as beliefs, goals and plans. It provides a sound architecture and framework for programming goal-oriented agents using established technologies like XML and Java. Goal-oriented protocols are a further step towards this aim, allowing to abstract away from low-level message passing.

4.1 Realization Approach

The domain interaction analysis process allows deriving generic interaction points from AUML protocol descriptions, which are described in terms of goals. During protocol execution, these goals have to be handled or posted from protocol-specifc but domain-independent agent behavior. In BDI agent systems such as Jadex, JACK or Jason (see [3]) such behavior can be captured in generic plans which have to be written once, and can be reused in different applications employing the same protocols. A problem with this approach is that plans are not sufficiently expressive for representing self-contained functionalities.

Hence, Jadex implements the extended capability concept [5], which allows to capture BDI-specific agent functionality as a reusable module. Capabilities group together functionally related beliefs, goals, and plans and exhibit a clearly defined interface of accessible beliefs or goals. To support the development of agents based on goal-oriented protocols, a so called *Protocols capability* has been realized as part of the current Jadex release. Based on the derived interaction goals generic plans for standardized FIPA protocols[1] such as Request, (Iterated)ContractNet, as well as English- and Dutch-Auctions have been implemented. While those plans are encapsulated inside the capability, the capability exposes the necessary goals needed to control the protocol execution.

4.2 Example Protocol: Goal-Oriented Contract-Net

Fig. 2 shows the result of the domain interaction analysis for the contract-net protocol [12]. Four different goals have been identified (two for each role). On the initiator side the *achieve cnp_ initiate* goal states that a task should be delegated using a contract-net negotiation. It has mandatory in-parameters for the call-for-proposal description (in-parameter *cfp*) and the potential participants (in-parameter *receivers*) and an optional parameter for additional local information

[1] see `http://www.fipa.org`

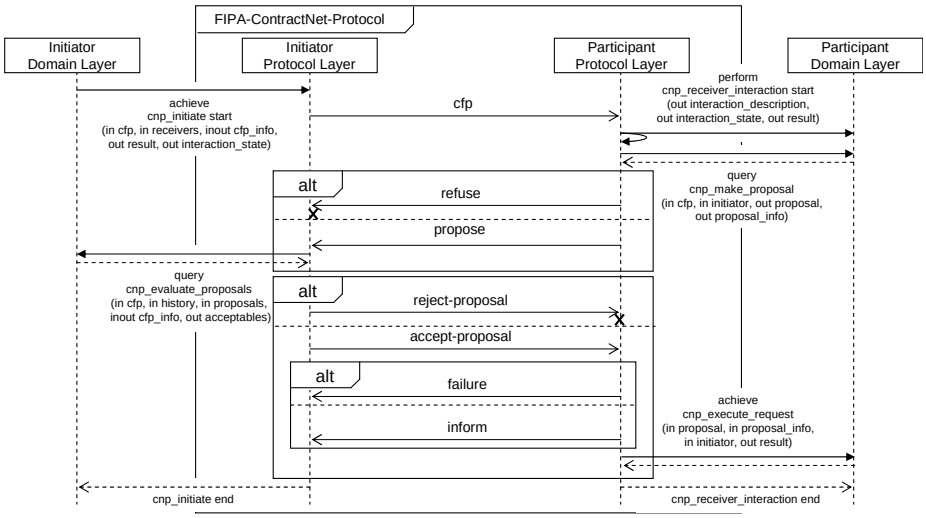

Fig. 2. Goal-oriented AUML contract-net specification (based on [12])

(in-parameter *cfp_info*). When the goal finishes (*cnp_initiate* end) the result of the negotiation is contained in the out-parameter *result*.

During the automatic protocol execution three interaction points have been identified at which the protocol layers need to initiate domain activities. First the initiator side sends the cfp-description to all potential participants and waits for their replies. On the participant side the *query cnp_make_proposal* goal is used to retrieve a proposal (out-parameter *proposal*) for the received cfp-description (in-parameter *cfp*). Additional information to the proposal itself can be stored in the optional out-parameter *proposal_info*. The proposal (or a refuse message in case no proposal was retrieved) will be automatically sent back to the initiator side. The initiator side uses the *query cnp_evaluate_proposals* goal to let the domain layer determine which of the received proposals (in-parameter *proposals*) should be accepted (out-parameter *winners*). For this decision it may use the original cfp and cfp_info values. The gained information is used to automatically send accept resp. reject messages to the participant side. Every participant whose proposal has been accepted utilizes the *achieve cnp_execute_request* goal to instruct the domain layer to execute the given task (in-parameter *task*). The result of the execution (out-parameter *result*) will be transferred back to the initiator side and will be stored in the original *cnp_initiate* goal.

4.3 Goal-Oriented Contract-Net Implementation

Relevant parts of the contract-net implementation are shown in Fig. 3. The interface of the capability mainly consists of the goals derived from the interaction analysis discussed beforehand (lines 7-17). The protocol layer is realized by the plans encapsulated within the capabilty (lines 18-23). Exemplarily, the specification of the *cnp_receiver_plan* is shown. This plan will be created in response to the receipt of a call-for-proposal message (*cnp_cfp*, line 21) and implements the

```
1  <capability name="Protocols" package="jadex.planlib">
2    <beliefs>
3      <belief name="cnp_filter" class="IFilter" exported="true">
4        <fact>IFilter.NEVER</fact>
5      </belief> <!-- Other beliefs omitted for brevity. -->
6    </beliefs>
7    <goals>
8      <achievegoal name="cnp_initiate" exported="true">
9        <parameter name="cfp" class="Object"/>
10       <parameter name="cfp_info" class="Object"/>
11       <parameterset name="receivers" class="AgentIdentifier"/>
12       <parameterset name="result" class="Object" direction="out"/>
13     </achievegoal>
14     <querygoal name="cnp_evaluate_proposals" exported="true">...</querygoal>
15     <querygoal name="cnp_make_proposal" exported="true">...</querygoal>
16     <achievegoal name="cnp_execute_task" exported="true">...</achievegoal>
17   </goals>
18   <plans>
19     <plan name="cnp_receiver_plan">
20       <body>new CNPReceiverPlan()</body>
21       <trigger><messageevent ref="cnp_cfp"/></trigger>
22     </plan> <!-- Other plans omitted for brevity. -->
23   <plans>
24   <events>
25     <messageevent name="cnp_cfp" type="fipa" exported="true">
26       <parameter name="protocol" class="String" direction="fixed">...</parameter>
27       <parameter name="performative" class="String" direction="fixed">
28         <value>SFipa.CFP</value>
29       </parameter> <!-- Other parameters omitted for brevity. -->
30       <match>$beliefbase.cnp_filter.filter ($messagemap)</match>
31     </messageevent> <!-- Other events omitted for brevity. -->
32   </events>
33 </capability>
```

Fig. 3. Cutout of the Protocols.capability.xml

participant role of the protocol. The actual code of the plan is contained in the Java CNPReceiverPlan class (cf. line 20) and not shown here. For message handling the plans make use of predefined message types (lines 24-32) derived from the FIPA protocol specifications. As an example, the call-for-proposal message (*cnp_cfp*, lines 25-31) is further illustrated. Besides some parameter specifications, e.g. for performative and protocol type (lines 26-29), this message contains a match expression (line 30), for configuration purposes as described below.

Agents might contain several protocol capabilities for different purposes and need to decide in which capability an incoming message should be processed. Therefore, the *cnp_filter* belief (lines 3-5) provides a filter used within the match expression. The belief (which turns off the participant role using IFilter.NEVER) can be overridden for specifying which calls for proposals should be handled.

5 Example Application

To illustrate how the goal-oriented approach can be used in practice the book-trading scenario from [2] is used, where personal buyer and seller agents are responsible for trading goods such as books according to instructions given by their principals. The market-based coordination strategy follows the contract-net

```
1  <agent name="Buyer" ...>
2    <goals>
3      <achievegoal name="purchase_book" recur="true" recurdelay="10000">
4        <parameter name="order" class="Order"/>
5        <targetcondition>Order.DONE.equals($goal.order.getState())</targetcondition>
6        <failurecondition >$beliefbase.time > $goal.order.getDeadline().getTime()</failurecondition>
7      </achievegoal>
8      <achievegoalref name="df_search"><concrete ref="dfcap.df_search"/></achievegoalref>
9      <achievegoalref name="cnp_initiate"><concrete ref="procap.cnp_initiate"/></achievegoalref>
10     <querygoal name="cnp_evaluate_proposals">
11       <assignto ref="procap.cnp_evaluate_proposals"/>
12       <parameterset name="winners" class="Object" direction="out">
13         <values evaluationmode="dynamic">
14           new Object[]{ select one Integer $price from $goal.proposals
15             where ((Order)$goal.cfp_info).getAcceptablePrice() >= $price.intValue()
16             order by $price }
17         </values>
18       </parameterset> <!-- Other parameters omitted for brevity. -->
19     </querygoal>
20   </goals>
21   <plans>
22     <plan name="purchase_book_plan">
23       <parameter name="order" class="Order">
24         <goalmapping ref="purchase_book.order"/></parameter>
25       <body>new PurchaseBookPlan()</body>
26       <trigger><goal ref="purchase_book"/></trigger>
27     </plan>
28   </plans>
29 </agent> <!-- Other elements omitted for brevity. -->
```

Fig. 4. ADF excerpt of the buyer agent

protocol, equally respecting the goals of buyer and seller agents. It is assumed that buyers take the initiator role of the protocol while sellers play the responder role. The goal-oriented implementation of the booktrading example is part of the Jadex distribution and is divided into files specific to the buyer resp. seller agent, as well as common files (e.g. ontology and GUI classes). Both agents store Order objects in their beliefbase, which represent the current buy or sell orders entered by the agents principals through the user interface of each agent.

5.1 Buyer Agent Implementation

In Jadex, an agent type is described by a so called agent definition file (ADF). Important parts of the buyer agent ADF are shown in Fig. 4. Instances of the *purchase_book* goal (lines 3-7) are created when new orders are added through the user interface. To be continuously retried whenever it fails, the goal has a recurdelay of 10 seconds (line 3). For holding the Order object entered through the GUI, the goal has one parameter *order* (line 4). In the target condition, the goal is considered to be reached, when the order is done, i.e. the desired book was successfully bought (line 5). When the book could not be obtained before the order deadline, the goal fails (line 6). To search for agents providing specific services and to initiate a contract-net interaction, the *df_search* goal (line 8) and the *cnp_initiate* goal (line 9) are included. During the execution of the contract-net interaction, which is performed inside the generic protocols capability, an instance of the *cnp_evaluate_proposals* goal (lines 10-19) is posted, when all

```
1  public void body() {
2     IGoal df_search = createGoal("df_search");
3     df_search.getParameter("description "). setValue(getPropertybase(). getProperty("service_seller "));
4     dispatchSubgoalAndWait(df_search);
5     AgentDescription[] result = (AgentDescription[]) df_search.getParameterSet("result "). getValues();
6     if (result . length == 0) fail();
7     AgentIdentifier [] sellers = new AgentIdentifier[result . length];
8     for (int i = 0; i < result . length; i++)
9        sellers [i] = result[i]. getName();
10
11    Order order = (Order)getParameter("order").getValue();
12    IGoal cnp = createGoal("cnp_initiate ");
13    cnp.getParameter("content").setValue(order . getTitle ());
14    cnp.getParameterSet("receivers "). addValues(sellers );
15    dispatchSubgoalAndWait(cnp);
16
17    order . setExecutionPrice ((Integer )(cnp.getParameterSet("result "). getValues()[0]));
18    order . setExecutionDate(new Date());
19 }
```

Fig. 5. The purchase book plan of the buyer

proposals have been collected and need to be rated against each other. In the booktrading domain, the buyer agent compares the prices of the proposals to the acceptable price as given in the order from the user (lines 14-16). When no acceptable proposal is present, the query goal automatically fails due to an empty *winners* parameter set, otherwise the interaction will terminate with the buyer accepting the cheapest proposal (due to ordering defined in line 16). In the plans section (lines 21-28), the *purchase_book_plan* is defined (line 22-27), which is triggered by the *purchase_book* goal (line 26). The *order* parameter from the goal is mapped to a plan parameter (lines 23-24), while the body tag (line 25) refers to the Java class implementing the plan.

The body of the *purchase_book_plan* is shown in Fig. 5. It contains two main parts: First, it has to determine negotiation partners using a *df_search* subgoal (lines 2-9). In a second step a parallel negotiation with all suitable sellers is performed represented by the *cnp_initiate* subgoal (lines 11-15). When no error occurs during the negotiation (in which case the plan would immediately fail and exit), the result is finally stored in the Order object (lines 17-18) making the goal succeed due to its target condition. When the goal is aborted before the plan finishes (e.g. if the deadline passes during an ongoing interaction), the plan and its subgoals will also be aborted, in which case the interaction is automatically terminated using the standardized FIPA-Cancel-Meta-Protocol (cf. [12]).

5.2 Seller Agent Implementation

Domain activities of the seller are triggered by the generic goals of the protocols capability, which are included as shown in Fig. 6. When a call-for-proposal message is received, the *cnp_make_proposal* goal (lines 3-19) is created automatically, allowing the agent to decide about making an offer. This query goal is defined declaratively by specifying directly the out-parameter values (lines 6-18), hence no plan is necessary to handle the goal. Instead, the current beliefs of the

```
1  <agent name="Seller" ...>
2    <goals>
3      <querygoal name="cnp_make_proposal">
4        <assignto ref="procap.cnp_make_proposal"/>
5        <parameter name="cfp" class="Object">...</parameter>
6        <parameter name="proposal_info" class="Object" direction="out" optional="true">
7          <value evaluationmode="dynamic">
8            select one Order $order from $beliefbase.orders
9            where $order.getTitle().equals($cfp) && $order.getState().equals(Order.OPEN)
10           order by ($beliefbase.time - $order.getStartTime())
11             / ($order.getDeadline().getTime()-$order.getStartTime())
12         </value>
13       </parameter>
14       <parameter name="proposal" class="Object" direction="out">
15         <value evaluationmode="dynamic">
16           ((Order)$goal.proposal_info).getAcceptablePrice()
17         </value>
18       </parameter>
19     </querygoal>
20     <achievegoalref name="cnp_execute_task">
21       <concrete ref="procap.cnp_execute_task"/></achievegoalref>
22   </goals> <!-- Plans and other elements omitted for brevity. -->
23 </agent>
```

Fig. 6. ADF excerpt of the seller agent

agent are checked, and if the agent currently whishes to sell the requested book (identified by the title in line 9), the acceptable price (line 16) is returned as a proposal. When the buyer accepts the proposal, a *cnp_execute_task* goal is created to complete the transaction. This goal is handled by an *execute_order_plan* (not shown), which may handle delivery and payment issues.

The example shows the clean separation of protocol execution and domain activities, letting application developers focus on domain behavior. Moreover, architectural concepts such as goals and plans can be used as usual also for implementing interaction behavior. Finally, the code is more simple compared to a functionally equivalent implementation of the booktrading scenario in JADE as described in [2]. Although the JADE implementation uses generic classes as well for the contract-net implementation, the buyer and seller implementations are 30-50% larger than the corresponding Jadex implementations presented here.

6 Summary and Outlook

This paper tackles questions concerning the interaction of agents in a multi-agent system, and focuses on the mirco level of interactions, i.e. how to describe and implement interaction protocols from the viewpoint of single agents. Central questions are how to derive and accurately represent the individual objectives in the course of an interaction and how to relate interactions to the agent architecture and to domain-specific behavior. A review of related work reveals that existing approaches do not offer a unified domain-centric view to all these questions, and instead mostly focus on concrete message sequences.

Based on these findings, a new approach is proposed, which brings together an abstract BDI-centered view on domain activitities with predefined interaction

protocols. As result *goal-oriented interaction protocols* are derived leading to a reduced effort for realizing agent communications. Advantages of the approach are that interaction objectives are conserved in the implementation and a tight integration into the internal agent architecture is achieved. Moreover, the domain layer is separated from the protocol layer facilitating understandability, maintainability and reusability of code. A generic realization and an example application have been presented, demonstrating the feasibility of the approach. Future work can be undertaken in areas such as dynamic protocol selection or execution. E.g., a protocol engine would allow executing abstract user defined protocols additionally to the standardized protocols of the Protocols capability.

References

1. B. Bauer, J. Müller, and J. Odell. Agent UML: A formalism for specifying multiagent software systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):207–230, 2001.
2. F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent systems with JADE*. John Wiley & Sons, 2007.
3. R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni. *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, 2005.
4. M. Bratman. *Intention, Plans, and Practical Reason*. Harvard Press, 1987.
5. L. Braubach, A. Pokahr, and W. Lamersdorf. Extending the Capability Concept for Flexible BDI Agent Modularization. In *3rd Int. Workshop on Programming Multiagent Systems (ProMAS 2005)*, pages 139–155. Springer, 2006.
6. L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf. Goal Representation for BDI Agent Systems. In *2nd Int. Workshop on Programming Multiagent Systems (ProMAS 2004)*, pages 44–65. Springer, 2005.
7. L. Cabac and D. Moldt. Formal semantics for AUML agent interaction protocol diagrams. In *Agent-Oriented Software Engineering (AOSE 2004)*, p. 47–61, 2005.
8. C. Cheong and M. Winikoff. Hermes: Designing goal-oriented agent interactions. In *Agent-Oriented Software Engineering (AOSE 2005)*. Springer, 2005.
9. M. Dinkloh and J. Nimis. A tool for integrated design and implementation of conversations in multiagent systems. In *1st Int. Workshop on Programming Multi-Agent Systems (ProMAS 2003)*, pages 187–200. Springer, 2004.
10. L. Ehrler and S. Cranefield. Executing agent UML diagrams. In *Autonomous Agents and Multi-Agent Systems (AAMAS 2004)*, pages 906–913. IEEE, 2004.
11. J. Ferber. *Multi-Agents Systems - An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, 1999.
12. Foundation for Intelligent Physical Agents (FIPA). *FIPA Contract Net Interaction Protocol Specification*, December 2002. Document no. FIPA00029.
13. P. Giorgini, M. Kolp, J. Mylopoulos, and M. Pistore. The Tropos Methodology. In *Methodologies and Software Engineering for Agent Systems*. Kluwer, 2004.
14. J. McCarthy. Ascribing mental qualities to machines. In *Philosophical Perspectives in Artificial Intelligence*, pages 161–195. Humanities Press, 1979.
15. A. Pokahr, L. Braubach, and W.Lamersdorf. Jadex: A BDI Reasoning Engine. [3].
16. C. Rooney, R. Collier, G. O'Hare. Viper: A visual protocol editor. In *Coordination Models and Languages (Coordination 2004)*, pages 279–293. Springer, 2004.
17. A. Scheibe. Ausführungsumgebung für FIPA Interaktionsprotokolle am Beispiel von Jadex. Diplomarbeit, University of Hamburg, 2003. (in German).

18. A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *Requirements Engineering (RE 2001)*, pages 249–263. IEEE Press, 2001.
19. Whitestein Technologies. *Semantic Communication User Manual*, LS/TS Release 2.0.0 edition, 2006.
20. M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2001.