# Reusable Interaction Protocols for Workflows

Alexander Pokahr and Lars Braubach

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

**Abstract.** With the advent of collaborative business processes that may include different parties, the interaction means between those processes steadily gains more importance. Despite this issue the modeling of message-based interactions is very complex and error-prone due to the inherent properties of asynchronous communication. In order to alleviate this problem several predefined interaction protocols such as contract-net or different auction types have been defined in the context of multi-agent systems. In this paper it is shown how these interaction protocols can be described in a reusable way for BPMN workflows. The underlying idea is to hide the complete message-based complexity of the underlying protocol specification and allow to employ them in a completely domain oriented fashion, i.e. the workflow modeler is only concerned with initiating a specific type of interaction and fetching the results. For this purpose the ideas of agent-based goal-oriented interactions have been simplified and transferred to the workflow modeling area. The usefulness of the approach is exemplified by an example that illustrates how contract-net negotiations can be set-up.

## 1 Introduction

Business processes typically describe activity sequences for bringing about business objectives. When business processes get more complex and several autonomous partners are required for producing a desired process outcome, the collaboration perspective between those participating partners gains importance. This collaboration perspective highlights different aspects that are distinct from the internal process description view and include factors like decentralized process management, peer-to-peer interactions and negotiations [LCV09]. A collaboration can be seen as an interaction between autonomous entities, whereby the corresponding perspective has to deal with properties and rules guiding this interaction. Because of the different perspective that collaborative processes emphasize, additional modeling approaches for describing coordination between processes have been developed (e.g. WS-CDL [W3C04], UML2 sequence diagrams [OMG05]). Typically, these approaches take an interaction protocol stance, which focuses on the allowed sequences of messages between the involved parties. Automated execution of collaborative business processes requires that interactions can be adequately modeled and also be equipped with sufficient technical details. The mechanism design itself and thus the modeling of interaction protocols is a very complex and challenging task

[Woo01] so that several general purpose interaction protocols have been developed. These prebuilt protocols are typically domain independent and target e.g. areas like auctions and negotiations. Due to the complexity of interaction protocols it would be beneficial to be able to make use of such predefined protocols and employ them as reusable modules in the context of another business process. This requires not only interactions being described in a technically interpretable manner but also that the interconnection between message flow and domain behavior is made explicit.

In this paper an approach is presented that facilitates the usage of predefined interaction protocols and shows how these protocols can be integrated into domain dependent processes. Despite the advantages of the approach, it has to be noted that not all kinds of interactions can be modeled using such prebuilt protocols, so that manual design still may be necessary depending on the concrete use case at hand. In Section 2 related work regarding collaborative business processes is discussed by specifically highlighting approaches that focus on process interaction modeling and execution. Section 3 introduces the concepts of reusable workflow protocols and illustrates patterns for mapping interaction protocols to workflows. Thereafter, Section 4 exemplifies the realization of reusable protocols and presents the Jadex execution infrastructure. The paper concludes with a summary and an outlook in Section 5.

## 2 Related Work

Collaborative workflows (also named inter-organizational workflows) have been subject of intensive research in different areas of computer science. Common ground of most research work is the usage of interaction protocols for describing the public parts of such processes. In the following it will be especially discussed how interaction protocols have been exploited for modeling and execution in multi-agent systems and which approaches in the workflow community exist for integrating such protocol specifications.

In multi-agent systems modeling and execution of interaction protocols has attracted much attention since the beginnings of the field, as agents communicate solely in a message-oriented asynchronous way. In the context of the Foundation for Intelligent Physical Agents[1] (FIPA), which is a subdivision of IEEE responsible for standards in the agent area, AUML (Agent UML) sequence diagrams have been developed for representing message exchanges between different roles of a protocol including message control flow elements like branches or loops. These representations have strongly influenced the evolution of UML2 sequence diagrams, which are a de-facto standard for modeling interaction protocols nowadays. One main idea that can be found in many agent-based approaches [DN04,Hug02,PTW07] is automatic code generation starting from interaction protocols. Typically, agent behavior skeletons are produced, which reflect the message flow via method signatures and need to be refined by a programmer, who has to add custom code for the required domain logic. One fundamental problem inherent to all such approaches is the *post-editing problem* [Sze96], which denotes that handcrafted code will be lost whenever code generation has to be performed again, e.g. due to a

[1] www.fipa.com

changed protocol specification. The most prominent way to address this problem, is by employing the model driven architecture (MDA) instead of simple code generators. MDA then takes care that a clear separation of the different model abstraction layers is preserved and regeneration of code (even targeted at different implementation platforms) is always possible. An alternative that was e.g. used in [EC04] consists in using an AUML interpreter, which is able to process corresponding specifications at runtime. In this case the connection of protocols with domain logic becomes crucial. It can be solved by explicitly introducing domain interaction points based on message receival. In order to make such protocol domain interface minimal in [BP07] it was suggested to base the interface not on received messages but only on the underlying domain actions required at specific decision points in the protocol.

The workflow community has tackled the modeling and realization of collaborative workflows in many cases by using code generation. In contrast to the agent approaches discussed above mainly the target representation is different, whereby as source representation in many cases also (modified) UML2 sequence diagrams [vdAW01,VLRC10] or semantically similar representations like WS-CDL [W3C04] have been used. As target languages especially petrinets [vdAW01,FÁBE06] and WS-BPEL [VLRC10] have been utilized. Similarly, also MDA has been applied in the more advanced solutions for avoiding the post-editing problem and for being able to generate different execution languages, e.g. pertri-nets for verification and WS-BPEL for operation. Besides these generation approaches, also an alternative manual development technique has been proposed. It relies on the idea that in a first step the choreography of business processes should be modeled e.g. in [DB08] via iBPMN (bpmn for interactions) and in [vdAW01] via sequence diagrams and petri-nets. Based on this description the public interfaces for private processes should be defined. Private processes have to implement the defined communication points in according to the global collaboration view. This technique is reasonable but does not take into account any automation aspects for rapid system development.

In summary it can be stated that nearly all approaches make use of code generation mechanisms that allow starting with a high-level interaction based choreography description and generating individual code snippets for all roles of the protocol. Despite that MDA alleviates the post-editing problem of generator based approaches and is very versatile, the mechanism still requires individual protocol definition and customization in each application case. In contrast, in this paper we aim at an interpreter based solution, which can directly execute choreography specifications and allows established and domain independent interaction protocols being included in workflows as reusable modules. For this purpose especially the clean separation of protocol and domain logic is of importance to allow users including and invoking prebuilt interaction protocol workflows.

## 3 Concepts

In this section the concepts for modeling reusable interaction protocol workflows are described. First, the general approach will be presented followed by specific patterns for mapping UML-based protocol specifications to BPMN processes.

### 3.1 General Approach

Many interesting interaction protocols have been standardized in the area of agent systems, e.g. contract net negotiation as well as Dutch and English auctions [FIP02a,FIP02b,FIP02c]. The approach presented in this paper aims at providing reusable implementations of these established protocols. Reusable protocol implementations reduce the complexity of implementing distributed systems and further increase robustness, due to relying on well-tested solutions. The approach continues the work from [BP07] and transfers the agent ideas to workflow-based systems.

The basic idea is to map the interaction protocol to a process, which represents the choreography of possible message exchanges. The choreography is described using the business process modeling notation (BPMN), which allows to capture all roles of the interaction in a single process specification. To employ this process as a reusable protocol implementation it has to be made sure, that each of the roles can be independently executed. Therefore the pure choreography has to be extended with corresponding decision points, which capture the domain logic. The decision logic is modeled as abstract subprocesses. When a protocol is reused in a specific setting, the developer only needs to state the role to be executed and provide reference to the concrete processes implementing the domain logic.

Previous work has already started addressing the issue of mapping AUML interaction protocols to executable process descriptions (e.g. [LCV09]). This paper builds upon existing work and extends it with a special focus on reusability of process descriptions. Therefore when mapping an interaction protocol to a process description, the process must not contain any domain specific logic (e.g. concrete branching conditions). Yet, it must provide clear extension points where domain logic can be seamlessly integrated.

### 3.2 Patterns

The activity of deriving a process description from an interaction diagram is guided by patterns that illustrate how to map common message structures into appropriate process structures. Our work extends patterns that can be found in e.g. [LCV09]. Main focus of the patterns is the reusability aspect, i.e. how to integrate domain logic in a non-invasive fashion.

The first pattern represents a choice point in one of the protocol roles. In the interaction diagram (see Figure 1, left), the choice manifests in one of two different messages that may be sent. Yet, the reason for the choice is not apparent in the diagram, because it belongs to the private behavior of the sender. In order to decide about the subsequent protocol flow, the sender may need to execute arbitrary complex business logic. Therefore, the BPMN mapping (Figure 1, right) introduces a subprocess *Prepare Decision*. It is abstract and needs to be mapped to appropriate domain logic. The concrete process to execute for *Prepare Decision* can be specified as a configuration option and does not require the BPMN protocol process to be changed for different application use cases.

The second pattern (Figure 2) is a specialization of the first pattern and represents a common use case that can be found in many interaction protocols. In this pattern, the decision

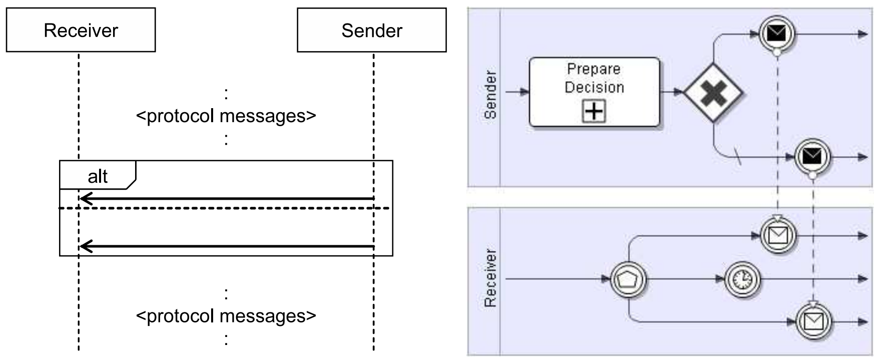


**Fig. 1.** Alternative pattern in UML (left) and BPMN (right)

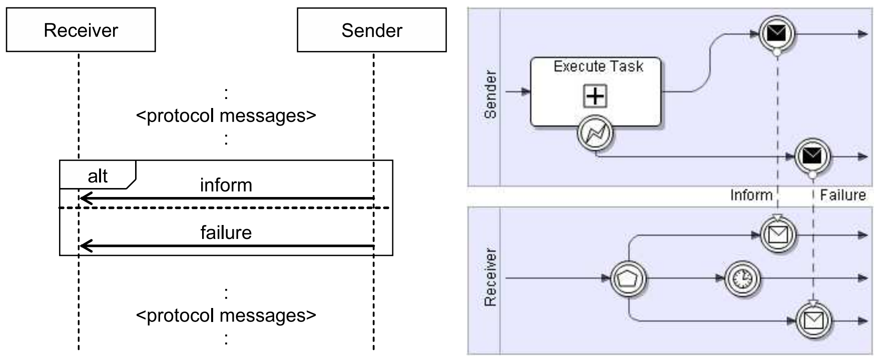


**Fig. 2.** Task execution pattern in UML (left) and BPMN (right)

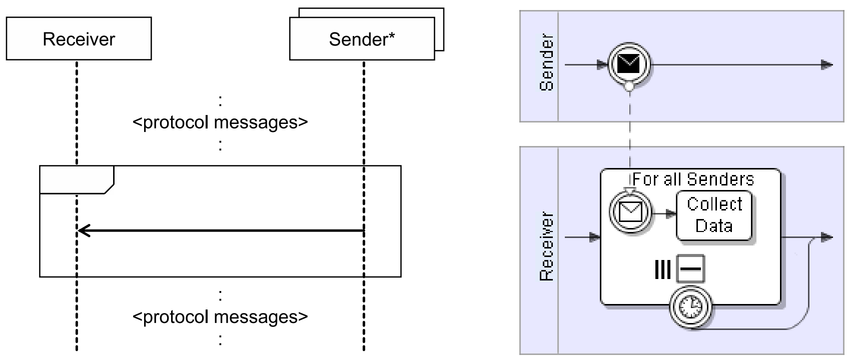


**Fig. 3.** Multiple receive pattern in UML (left) and BPMN (right)

is not between arbitrary messages but between a *failure* and an *inform* message. This can be interpreted as a communication of the result of a task execution at the sender side. If the task succeeds, the receiver is informed about the result. If the task fails, the failure is communicated to the receiver. Therefore in the BPMN mapping, the prepare decision task and corresponding gateway from the first pattern are replaced by an *Execute Task* subprocess and an exception handler. The advantage of using the second pattern instead of the first is that the domain logic only has to deal with the task itself. While in the first pattern an explicit decision about failure or success has to be made, the second pattern handles the failure case automatically.

In complex interaction protocols there is usually more than one process instance playing a particular role. E.g. in an auction setting, one auctioneer interacts with arbitrary many bidders. Special care has to be taken for mapping this potential multiplicity to appropriate process structures. In the third pattern (see Figure 3) one receiver expects a message of each of potentially many protocol participants. Therefore, while the sender side only interacts with one receiver, the receiver side has to deal with many senders. In BPMN this is captured by a parallel subprocess at the receiver side. The content of the subprocess is executed for each expected sender. The pattern requires that the concrete participants of the interaction are known at this place in the protocol. One question to be answered in the mapping is the scope of the parallel activity. In the UML diagram (Figure 3, left) a compartment is used to define the subset of the protocol where the multiple sending happens. When this compartment ends (i.e. once all messages are received), the receiver continues to execute sequentially (e.g. looking at all received bids and selecting one). In BPMN a generic task is introduced to collect the contents of the single received messages. The combined result is made available when the parallel process ends. Because the task only collects data and does not analyze it, no domain specific functionality is required.

All three patterns introduce timeouts for message receival. For open systems where interacting participants may disappear independently, timeouts are essential for robust execution and dealing with partial failures. In the first two patterns, the timeouts are modeled using intermediate time events that may occur as alternative to a message receival. In the third pattern there is only one timeout handler for the parallel subprocess receiving multiple messages. The actual timeout values are not hard-coded into the reusable protocol processes, but can be configured when using the protocol in a specific application context. Therefore the protocols can be used in human-centered processes (timeouts e.g. several days) as well as automated processes (timeouts of a few seconds).

## 4 Realization

To illustrate the presented concepts this section presents a well-known example interaction protocol and its corresponding mapping to BPMN. Furthermore, the Jadex Active Components infrastructure [PBJ10] is introduced as a modeling and runtime environment for executing the modeled process.

## 4.1 Example: Contract Net

The contract net protocol is an established interaction protocol for task allocation [Smi80]. The initiator of the protocol seeks to delegate a task to one or more subcontractors. The subcontractors can make individual proposals (e.g. concerning the price or quality of the task execution) and the initiator selects one or more of these proposals to be executed. The protocol, as standardized by FIPA, has been used in many different application areas and is therefore an interesting candidate for a protocol to be available as an easily reusable implementation.

Figure 4 shows the protocol in UML2 sequence diagram notation. The mapping of the protocol to BPMN is based on the patterns introduced in the previous section. In the following, the mapping will be discussed with respect to the different sections of the protocol. The first section of the protocol comprises the initial *cfp* (call for proposals) message and the reply messages of the participants (either *propose* or *refuse*). This section combines two patterns. At the participant side, there is a decision to be made about proposing or refusing (pattern 1). At the initiator side, the process needs to receive messages from multiple participants (pattern 3). The resulting BPMN (see Figure 5) thus represents a mixture of both patterns. At the participant side, pattern 1 is applied and the abstract *Make Proposal* subprocess is introduced to be mapped to corresponding domain logic for deciding about participation in the negotiation. At the initiator side, a parallel subprocess handles the sending of the cfp to each participant and the receival of the respective reply. The protocol ends at this point, when all participants send a refuse or no proposal is received before the timeout. This is represented by the corresponding gateway and the *proposals.isEmpty()* condition in the initiator process. When a participant does not make proposal, i.e. when the condition *proposal!=null* does not hold, this specific participant process ends, while other participants can continue to negotiate with the initiator.

The second protocol section comprises the decision of the initiator about which proposals to accept and the communication of the decision to the participants. Therefore a combination of a decision (pattern 1) and multiple message sending (similar to pattern 3 with send instead of receive) is included in the process. The decision is performed in the global scope (i.e. not in parallel for each single participant), to allow the corresponding domain logic to reason about all proposals at once. The accepted proposals are stored for later use in the last protocol section. The participant side of this protocol section is rather simple. When the proposal is rejected or a timeout occurs, the participant process ends. Otherwise the participant continues with the last protocol section.

When a participant is informed about the acceptance of its proposal, it starts the execution of the corresponding domain task (*Execute Request*). When the execution fails, the failure reason is communicated, otherwise the result is sent to the initiator (pattern 2). The initiator waits until is has received the results of all task executions, which are collected in turn (pattern 3).

For easy reuse of the protocol in specific application cases, the extension points for the required domain logic are clearly defined. Figure 6 summarizes the extensions points along with important input and output values. The domain logic interfaces are based on the analysis in [BP07] where further details, such as additional input/output values can be found. The sys-

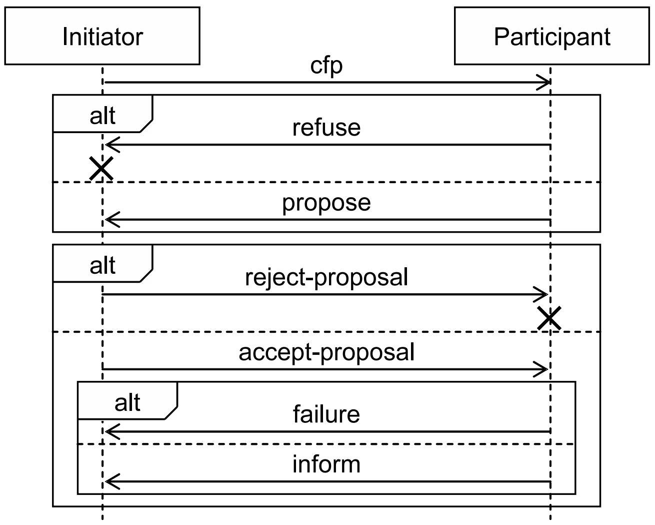


**Fig. 4.** The FIPA contract net protocol [FIP02a]

tem that wants to start the protocol instantiates the initiator process (extension point *Initiate*) and supplies a domain object representing the initial call for proposals (cfp) as well as a list of participants to include in the negotiation. After the protocol has completed the collected results of the task execution(s) are made available in the *result* output value. During the course of the protocol the evaluate proposal process (*epp*) needs to be executed. The concrete implementation to be used is already specified when initiating the protocol. When executed, the *Execute Proposals* subprocess receives as input all collected *proposals* and has to provide *proposal evaluations* as a result, stating which of the proposals should be accepted.

To participate in a negotiation, a system has to *Activate* the participant process (i.e. load and configure the process specification) by specifying the concrete subprocesses to be executed for the make proposal process (mpp) and the execute request process (erp). The process is instantiated whenever a matching cfp message is received from an initiator. The *Make Proposal* process is passed the initiator and the cfp object and can decide to make a *proposal*, which has to be supplied as result. If the participants proposal is selected by the initiator, the *Execute Request* process is started at the participant side. It receives the original proposal made and can supply some value as a *result* of the execution.

### 4.2 Jadex Active Components Infrastructure

The contract-net process has been implemented using the Jadex Active Component framework[2]. For describing processes in BPMN, the framework provides a modeling tool based

[2] `http://jadex.informatik.uni-hamburg.de/`

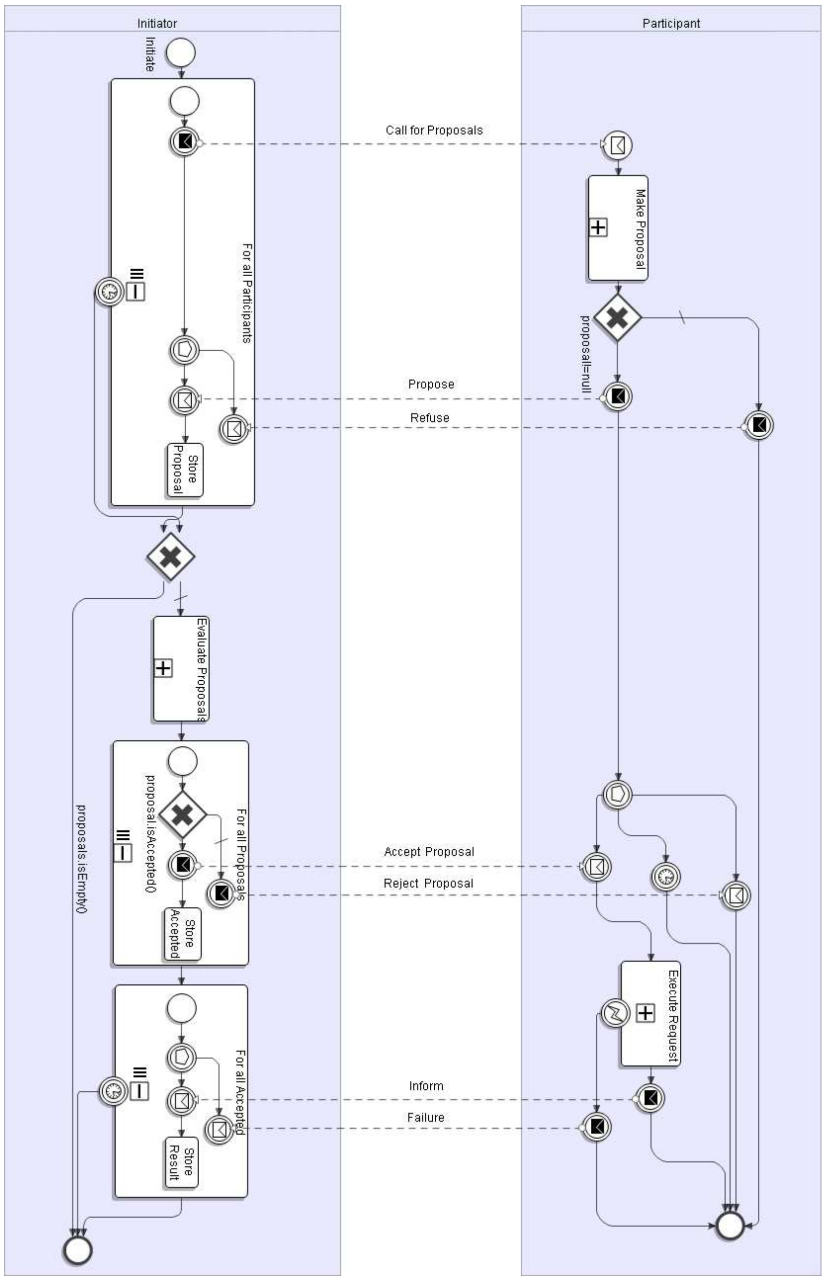


**Fig. 5.** The contract net protocol BPMN mapping

| Protocol Role | Extension Point | Input | Output |
|---|---|---|---|
| Initiator | Initiate | cfp, participants, epp | result |
| | Evaluate Proposals | proposals | proposal evaluations |
| Participant | Activate | mpp, erp | n/a |
| | Make Proposal | cfp, initiator | proposal |
| | Execute Request | proposal | result |

**Fig. 6.** Extension points for domain logic

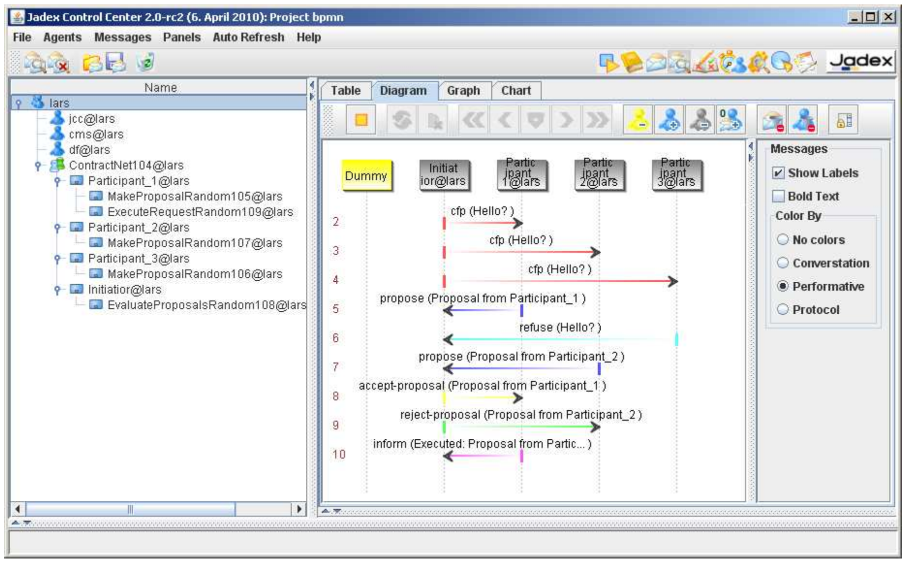
on existing eclipse tools. In addition a runtime platform and process interpreter allow modeled processes to be executed along with other types of components (e.g. agents) [PBJ10]. Additionally, the framework includes a number of runtime tools that allow debugging running components such as processes. In the context of protocol implementation, the most useful tool is the ComAnalyzer, which allows to monitor and visualize communication among processes. Recorded messages can be shown in different views (table, sequence diagram, 2D graph, bar/pie chart) and filtered according to developer rules.

Figure 7 shows a screenshot of the ComAnalyzer after it has recorded a contract net negotiation. The tree on the left shows the processes that have been executed. The diagram in the middle shows the sequence of messages exchanged between processes. For the domain logic, simple processes have been implemented, that perform random choices. It can be seen in the tree that the *MakeProposalsRandom* process has been started as a subprocess of each *Participant* process. In the diagram, one can observe that two participants have made a proposal, while *Participant_3* has sent a refuse. The *Initiator* process, shown at the bottom of the tree has started the *EvaluateProposalsRandom* process as a subprocess to evaluate the remaining two proposals. It can be seen in the diagram that the proposal from *Participant_1* is accepted while the other proposal is rejected. Therefore only Participant_1 starts the *ExecuteRequestRandom* subprocess, which leads to an inform message being sent back in case of success.

## 5 Summary and Outlook

Collaborative business processes require considering a global interaction perspective. This perspective highlights the roles of the different partners and defines their behavior according to the visible message sequences between them, i.e. a collaboration process is specified by peer-to-peer interactions of roles. Due to this peer-to-peer characteristics such processes are inherently decentralized and need to be executed in a cooperative fashion by the interaction participants.

In order to design and execute collaborative processes existing approaches mainly rely on code generation mechanisms that are versatile but also require customization of processes for each application case. In contrast, in this paper an interpreter centered approach has been proposed that is able to directly execute choreography specifications. This is currently achieved by modeling the choreography via BPMN according to established UML2 sequence diagram protocol specifications. Several patterns have been proposed in order to show how a systematic conversion from sequence diagrams to BPMN can be done.


Jadex Control Center 2.0-rc2 (6. April 2010): Project bpmn
File Agents Messages Panels Auto Refresh Help
Jadex
Name
Table Diagram Graph Chart
Dummy
Initiat ior@lars
Partic ipant 1@lars
Partic ipant 2@lars
Partic ipant 3@lars
Messages

```
lars
  jcc@lars
  cms@lars
  df@lars
  ContractNet104@lars
    Participant_1@lars
      MakeProposalRandom105@lars
      ExecuteRequestRandom109@lars
    Participant_2@lars
      MakeProposalRandom107@lars
    Participant_3@lars
      MakeProposalRandom106@lars
    Initiatior@lars
      EvaluateProposalsRandom108@lars
```

```
2   cfp (Hello? )
3   cfp (Hello? )
4   cfp (Hello? )
5   propose (Proposal from Participant_1 )
6   refuse (Hello? )
7   propose (Proposal from Participant_2 )
8   accept-proposal (Proposal from Participant_1 )
9   reject-proposal (Proposal from Participant_2 )
10  inform (Executed: Proposal from Partic... )
```

[✓] Show Labels
[ ] Bold Text
Color By: No colors / Converstation / [✓ Performative] / Protocol

**Fig. 7.** ComAnalyzer screenshot showing a contract net negotiation

In addition, one major objective of the presented approach is to allow reusing established interaction protocols like contract-net and different auctions. To achieve this, specific domain interfaces for protocols have been proposed, which allow for a customization of the domain behavior. Concretely this is achieved by introducing subprocess placeholders, which will be filled with domain helper workflows by the user and serve for delivering the required information for the protocol layer. Domain helper workflows need to be defined according to specific interfaces derived from the general domain interface of a protocol process.

It has been further shown that BPMN choreographies can be executed using the open source Jadex Active Components infrastructure, which supports agents as well as workflows and other component types. Predefined interaction protocols can be used as subprocesses within domain workflows and are instantiated and parameterized as subprocesses. One essential configuration aspect is the role that should be played as the whole choreography is modeled in one BPMN diagram. Future work will address building-up a library of ready-to-use protocols based on the FIPA interaction protocol specifications.

## References

[BP07] L. Braubach and A. Pokahr. Goal-oriented interaction protocols. In *5th German conference on Multi-Agent System Technologies (MATES 2007)*. Springer, 2007.

[DB08] G. Decker and A. P. Barros. Interaction modeling using bpmn. In *Business Process Management Workshops*, pages 208–219. Springer, 2008.

[DN04] M. Dinkloh and J. Nimis. A tool for integrated design and implementation of conversations in multiagent systems. In *Proc. of the 1st Int. Workshop on Programming Multi-Agent Systems (PROMAS 2003)*, pages 187–200. Springer, 2004.

[EC04] L. Ehrler and S. Cranefield. Executing agent UML diagrams. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004)*, pages 906–913. IEEE Computer Society, 2004.

[FÁBE06] J. Fabra, P. Álvarez, J. A. Bañares, and J. Ezpeleta. A framework for the development and execution of horizontal protocols in open bpm systems. In *4th Int. Conf. Business Process Management (BPM 2006)*, pages 209–224. Springer, 2006.

[FIP02a] Foundation for Intelligent Physical Agents (FIPA). *FIPA Contract Net Interaction Protocol Specification*, December 2002. Document no. FIPA00029.

[FIP02b] Foundation for Intelligent Physical Agents (FIPA). *FIPA Dutch Auction Interaction Protocol Specification*, December 2002. Document no. FIPA00032.

[FIP02c] Foundation for Intelligent Physical Agents (FIPA). *FIPA English Auction Interaction Protocol Specification*, December 2002. Document no. FIPA00031.

[Hug02] M.-P. Huget. Generating code for agent uml sequence diagrams. In *Proceedings of Agent Technology and Software Engineering (AgeS)*, Erfurt, Germany, 2002.

[LCV09] I. M. Lazarte, O. Chiotti, and P. D. Villarreal. Transforming collaborative process models into interface process models by applying an mda approach. In *9th Conf. on e-Business, e-Services and e-Society (I3E 2009)*. Springer, 2009.

[OMG05] Object Management Group (OMG). *UML Superstructure 2.0*, 2005.

[PBJ10] A. Pokahr, L. Braubach, and K. Jander. Unifying Agent and Component Concepts - Jadex Active Components. In *Proc. of MATES 2010*. Springer, 2010.

[PTW07] L. Padgham, J. Thangarajah, and M. Winikoff. Auml protocols and code generation in the prometheus design tool. In *Proc of the 6th Int. Conf. on Autonomous agents and multiagent systems (AAMAS'07)*, pages 1–2. ACM, 2007.

[Smi80] R. G. Smith. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Trans. on Comp.*, 29(12):1104–1113, 1980.

[Sze96] P. Szekely. Retrospective and challenges for model-based interface development. In *Design, Specification and Verification of Interactive Systems (DSV-IS 1996)*, pages 1–27. Springer, 1996.

[vdAW01] W. van der Aalst and M. Weske. The p2p approach to interorganizational workflows. In *Proc. of the 13th Int. Conf. on Advanced Information Systems Engineering (CAiSE'01)*, pages 140–156, London, UK, 2001. Springer-Verlag.

[VLRC10] P. D. Villarreal, I. Lazarte, J. Roa, and O. Chiotti. A modeling approach for collaborative business processes based on the up-colbpip language. In *Business Process Management Workshops*, pages 318–329. Springer-Verlag, 2010.

[W3C04] World Wide Web Consortium (W3C). *Web Services Choreography Description Language*, version 1.0 edition, December 2004.

[Woo01] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2001.