German Research Center for Artificial Intelligence (DFKI)

Multi-agent Communication for the Realization of Business-Processes

Thesis for obtaining the title of

Doctor of Engineering

of the Faculties of Natural Sciences and Technology

of **Saarland University**

by: Esteban León Soto

Supervisors: Prof. Dr. Jörg H. Siekmann and
Prof. Dr. Jörg P. Müller

Saarbrücken,
December 2012

Colloquium held on 20 September 2013

Dean of NTF I: Univ.-Prof. Dr. Mark Groves

Examination Board:

Reporters:
Prof. Dr. Jörg H. Siekmann and
Prof. Dr. Jörg P. Müller

Chairman:
Prof. Dr.-Ing. Philipp Slusallek

Research Assistant:
Dr. Klaus Fischer

Affidavit

I hereby swear in lieu of an oath that I have independently prepared this thesis and without using other aids than those stated. The data and concepts taken over from other sources or taken over indirectly are indicated citing the source. The thesis was not submitted so far either in Germany or in another country in the same or a similar form in a procedure for obtaining an academic title.

Saarbrücken, 30.11.2012

Aknowledgements

Bringing this thesis to successful conclusion has not been an easy task and it would not have been possible without the help and support of so many people to which I want to express my most sincere gratitude.

First of all, I want to thank my thesis supervisors for supporting me during the complete process, specially to Prof. Siekmann for his inspiring guidance and for dedicating so much effort to my work, in spite of the huge horizon of subjects and groups he has to conduct. My special gratitude for Prof. Müller, for providing so much input and guidance in so many aspects of this thesis, in spite of the distance, and for always dedicating his complete and sincere attention in every of the few occasions we managed to talk about my work. His feedback about the contents, but also about issues related to the process of doing a PhD, always helped me to keep the north in this long journey.

I want to give thanks to Saarstahl AG for providing such a vital support to our research and making so many interesting projects possible and for the cooperative spirit that has always been your characteristic.

I want to thank the Multi-agents group in DFKI for being so welcoming from the beginning and for the great years I spent working with you. The friendly atmosphere we, Cristián, Sven, Ingo, Christian, Stefan, Klaus and all the rest had, cannot be taken for granted. I want to specially thank Klaus Fischer, for making the whole experience possible, for creating such a friendly and sincere teamwork mood and also for always being there providing guidance and advice, even for the most intricate details of my work. For Cristián Madrigal, my companion in this long adventure, my deepest gratitude for his genuine friendship and almost endless patience without which, these whole years would not have had the great quality they had.

To all my friends in Germany: Adrian, Eduardo, Eric, Guillermo, Gustavo, Hansraj, Jasmina, Jorge, Josiane, Levi, Ralitza, Raúl, thank you so much for all the great times we had, for your support and always encouraging me to move on. My stay here would not have been so good without my friends in UFC-Wacker, thank you for helping me stay balanced physically and mentally.

To my loyal friends in Costa Rica: Alejandra, Andrés, Andrea, Carolina A., Carolina M., Diana, Gabriel, Karoline, Mario, Oscar and Róger, thank

you very much for waiting so long and never forgetting me. You always made me feel as if I had never left every time I came back and also motivated me to bring this phase to a happy ending.

My family has been my strongest support in this and any other goal I set in my life. My parents, Noemy and Humberto, have always done only their best to raise me and, in spite of the distance, never let me feel being away. My sisters, Margot and Carolina, have always supported me in any way imaginable, you have been great. Also my family by extension, Jürgen and Christine, thank you for taking such good care of me and treating me like one of you, your support and advice have been invaluable.

All of this would not have even started if it was not for Dana, my beloved wife. For your support, encouragement and fundamental motivation thank you very much indeed.

And last not least, thank you Daniel and Javier, you taught me what is worth in life. To you I dedicate this.

Abstract

As Internet and information technologies expand further into daily business activities, new solutions and techniques are required to cope with the growing complexity. One area that has gained attention is systems and organizations interoperability and Service Oriented Architectures (SOA). Web Services have grown as a preferred technology in this area. Although these techniques have proved to solve problems of low level integration of heterogeneous systems, there has been little advance at higher levels of integration like how to rule complex conversations between participants that are autonomous and cannot depend on some ruling or *orchestrating* system. Multi-agent technology has studied techniques for content-rich communication, negotiation, autonomous problem solving and conversation protocols. These techniques have solved some of the problems that emerge when integrating autonomous systems to perform complex business processes. The present research work intends to provide a solution for the realization of complex Business Process between heterogeneous autonomous participants using multi-agent technology. We developed an integration of Web Services and agent-based technologies along with a model for creating conversation protocols that respect the autonomy of participants. A modeling tool has been developed to create conversation protocols in a modular and reusable manner. BDI-Agents implementations that communicate over Web Services are automatically generated out of these models.

Zusammenfassung

Internet und Informationstechnik finden immer mehr Verwendung in alltäglichen Geschäftsaktivitäten und als Folge dessen, werden neue Lösungen und Verfahren gebraucht, um der steigenden Komplexität gerecht zu werden. Insbesondere Bereiche wie System- und Organizations- Interoperabilität, wie auch dienst-orientierte Architekturen (SOA) haben demzufolge mehr Aufmerksamkeit bekommen. Dabei sind Web Services zur bevorzugten Technologie geworden. Tatsächlich haben diese Techniken Probleme in niedrigeren Ebenen gelöst, die beim lntegrieren von heterogenen Systemen entstehen. Allerdings gab es bisher weniger Fortschritte in höheren Ebenen, wie der Regelung von komplexen Dialogen zwischen Teilnehmern, die aufgrund ihrer Autonomie, sich nicht nach anderen kontrollierenden oder orchestrierenden Systemen richten lassen. Multiagenten-Systeme haben Bereiche wie inhaltreiche Kommunikation, Handel, autonome Problemlösung und Interaktionsprotokolle im Detail geforscht. Diese Techniken haben Probleme gelöst, die beim Ausführen von komplexen Geschäftsprozessen auftreten. Die vorliegende Doktorarbeit beabsichtigt, mit Verwendung von Multiagenten-Technologien, eine Lösung für die Umsetzung von komplexen Geschäftsprozessen zwischen heterogenen autonomen Teilnehmern bereitzustellen. Wir haben eine Integrationslösung für Web Services und agenten-basierte Technologien zur Verfügung gestellt, zusammen mit einem Model für die Erstellung von Interaktions-Protokollen, die die Autonomie der Teilnehmer berücksichtigt. Ein Modellierungstool wurde entwickelt, um modulare und wiederverwendbare Interaktionsprotokolle gestalten zu können. Aus diesen Modellen kann man auch Implementierungen automatisch erzeugen lassen, welche BDI-Agenten, die über Web Services kommunizieren, verwenden.

Contents

List of Figures

Part I

Introduction

Web Services are a well accepted tool for interoperability and they have been used for integration of heterogeneous distributed systems. So far, they have been used primarily for *Remote Process Call* (RPC), a mature technique nowadays. Recently, Web Services have started to move towards agent–like models. Nick Jennings (Jennings 2001) claims that agents are one of the preferred ways for implementing distributed systems with a higher level of complexity, where communication and autonomy play an important role. As a matter of fact, distributed systems based on communication and autonomy are the objective for Web Services.

Hence, there is now increasing interest to integrate solutions for multi-agent systems and Web Services for several reasons: to create Service Oriented Architectures (SOAs) with goal oriented services that achieve them autonomously and that allow for wider integration of different systems (Barry 2003), to take advantage of planning capabilities of agents in a Web Services scenario, to provide advanced services, to use them as organizing tools for searching Web Services using semantic web techniques or for composing more complex services out of a set of existing elementary services.

The integration of Web Services and agent technologies has been difficult, because of their RPC tendency. RPC communication, mostly synchronous and dependent on the other participant, is in strong contrasts to the way agents communicate, asynchronous and autonomous. Now, new specifications are available that allow a more agent-like communication, for instance Web Service Addressing (W3C 2006*a*) (WS-Addressing) and Web Services Choreography Description Language (W3C 2005) (WSCDL). The slow but steady advance of these new paradigms are a concrete signal that the way systems are being integrated today is changing to a more complex communication between participants.

Business process implemented in computer systems reflect the actual processes of the organization and in most cases, production will be as agile as its business process allows (Weerawarana, Curbera, Leymann, Storey & f. Ferguson 2005), therefore they are of great value for organizations. For business processes, simple *message exchange patterns* (MEP) like those in RPC (request-response) quickly become very limited. Interactions involve longer more intricate processes even for a single transaction. A good example is the simple process of purchasing some good. Ignoring the phase of finding and choosing providers for a good, an order has to be placed, confirmed, delivered and payed for. This represents a process that indeed demands more than a simple request and a response. More correlated messages are exchanged and there are always different ways a message can be replied to, depending on the intentions of the participant.

Agent techniques for Web Services provide solutions for many complex problems related to SOAs: Semantic Web Services and agent-based reasoning provide better ways for discovering and ranking services from a mass of services. Also, for planning and composing complex services out of existing

services, discovery tools can be combined with planning tools to produce composed services or even *orchestrations*.

Business processes between autonomous organizations frequently manifest a set of properties, like autonomy observance or unnecessary discovery, because of pre-existent partnerships. There are many cases in which an organization wishes to work in partnership with another, normally well known, organization (or organizations), to jointly perform a complex business process. Even though partners wish to work together, they will want to stay autonomous after integration. In order to keep participant organizations autonomous, no central entity is acceptable, which is a common solution offered by software vendors. Orchestrations are not admissible, since they imply the dependence of organizations on the orchestrator entity, annulling autonomy.

Such a scenario demands integration of existing well known participants in such a way that each system can interact with the rest in order to achieve a common goal. The main requirement of such a solution is to rule how interaction is to be performed. It has to be as simple as possible, but at the same time, enable flexibility. Multi-agent systems have been doing this for a long time by using *interaction protocols*.

Interaction protocols define how participants are to behave. Similar to the rules in a game, interaction protocols specify how a complex interaction is to be performed and what the possible options are for a participant. Following the agreement represented by the protocol, each participant produces its interface in order to participate. This way, participants can be assured that, if all of them follow the rules as specified in the protocol, each of them will be capable of achieving what they intend. Interaction protocols are an important contribution for modeling and enactment of business processes.

In the present work, the issue of integrating Web Services and agent technologies will be approached again, this time taking into account the significant improvements in Web Services and integrating both technologies at the messaging level.

The Foundation for Intelligent Physical agents (FIPA) (FIPA 2002) is one of the preferred and more extensive agent specification techniques. The majority of investigations about agent communication are implemented using these specifications, some of them are relevant to the Service Oriented community, especially those for dynamic planning of services (composition of services), conversations patterns and protocols or autonomous negotiation. Apart of that, FIPA Interaction Protocols Specification (FIPA 2002*g*) is the only formal definition of generic peer-to-peer agent protocols (Shehory 2003).

Based on contributions from the multi-agent community, old and new, a consolidating model of interaction protocols will be proposed. Using this model, a tool for modeling protocols will be implemented. Models produced with this tool will be implemented by agents running on top of the Web Services integration tool.

These models and tools have been tested in a real scenario: a selection of business processes in the Saarstahl AG (Saarbücken, Saarland, Germany (Saarstahl AG n.d.)) will be modeled and realized using the tools and knowledge gained in the present work. Saarstahl has been chosen as test case, because of its complex production chain. Saarstahl is organized in different departments that, given their size, work autonomously. Each output shares different phases of production with many other products making the overview of the production chain a maze with frequent interposition of paths of different goods.

Chapter 1

Motivation

Integration of heterogeneous systems for business processes has been on top of the table for a long time. This work proposes a way for achieving integration using a paradigm based on multi-agent systems. This chapter explains the main concepts and motivates our goals.

1.1 Trend of cross-enterprise integration

Networks of computer systems enabled the connection of different parts of organizations and made the implementation of distributed business processes easier. Integration used well known techniques like file transfer, shared databases, remote procedure call (RPC) or messaging (Hohpe & Woolf 2003). This trend has worked well when it comes to integrate different parts of a single organization. Since these parts are governed by the same organization and since they pursue the same goals, integration has found less obstacles in its way. A more severe problem is the integration of parts of different organizations, what is known as business to business (*B2B*) integration. The architecture and implementation of a distributed system is more difficult, since there are different specifications and requirements of the organizations.

For this purpose, platform independent solutions and standards were proposed. Initially CORBA was one of the most common platform independent networking integration mechanism. Later Web Services appeared, serving the same purpose, this time using better adopted XML standards.

Web Services are a standards-based, XML-centric technology for the realization of SOAs (Weerawarana et al. 2005). Business processes are a reflection of the product a company produces, since they represent the course of production. Therefore there is a correspondence between the speed and capability to adapt and to create new business processes and the speed and capability to adapt and to create new products. Flexibility is necessary to prevail in dynamic markets and constantly changing environments which

often require ad-hoc solutions (Gibbins, Harris & Shadbolt 2003, van den Heuvel & Maamar 2003). Being capable of establishing, adapting and changing partnerships is a significant part of the necessary abilities organizations require. SOA and Web Services have emerged to suit the continuous evolution organizations are subject of (Barry 2003, Smith & Fingar 2002).

SOAs are a suitable way for an acceptable level of flexibility. Web Services make use of widely accepted XML-based standards. Using standards that are easy to adopt by different organizations is fundamental in order to achieve integration and to find a common ground where the requirements of different organizations can be conciliated.

One key driving force behind SOAs is the result of the constant evolution of business processes. Evolution means change and specifically, tasks, actors performing them and tools used to perform them change producing new scenarios that improve processes. It is in such a situation where new participants appear and practices like outsourcing come into play. Business processes are extended beyond single organizations and new *virtual enterprises* appear.

1.2 Business-process and automated negotiation

Communication styles, other than simple request-response interactions as usual in RPC, are necessary, in particular for business-processes. Very frequently even asynchronous communication will be preferred, where requesters work in a *send-and-forget* manner, not having to deal with connection problems, since messaging in this case can be delegated. By chaining many of these messages one after another, complex conversations can be built.

In the context of Service Oriented Computing (SOC) business process modeling and management is experiencing an increasing demand. Business Process are: the process that is performed by participants in order to achieve some business goal. A business process also used to be called a *workflow* and it comprises a description of the sequence and alternatives participants have in order to solve a given business problem, rules that apply to this process and any other condition or constraint that applies to each participant. Business processes provide a way to specify the logic of the SOA in terms of a process declarative dynamic behaviour, the way it changes over time as events occur (Havey 2005). In the transition from simple RPC to more complex interactions, techniques for modeling and managing these processes are necessary and especially in the area of B2B, Business Process modeling and management prove to be useful.

1.2.1 Automated negotiation

With Web Services it is possible to introduce capabilities to make decisions based on the information provided by partners and this way move beyond the simple information passing purpose of a distributed computer system (W3C n.d.). When such a capability is present on at least two participants in a business process, the whole process can be called an automated negotiation. Even though it is hard to imagine that a manager would be willing to delegate his decisions to some software entity in business practice and therefore one wouldn't expect automated negotiations to be common, there are still some situations where this is the case. An example are auctions: situations in which an entity desires to look for some resource or service and therefore convokes a set of participants to make offers. The initiator then chooses a proposal and proceed to make use of it. There are also other scenarios in business process where some decisions are actually to be automatized. Independent of the issue of what a system owner is ready to delegate to software representatives, a solution for implementing such a scenario is required to provide enough flexibility to give participants room for action.

For an organization to be able to implement and participate in automated negotiations with other external participants, not only do software entities need to be sophisticated enough, but also there must be a good interaction grounding for these participants to work on. A clear understanding of how a negotiation is to be undertaken, and what will be allowed to be done and what not has to be clearly represented. Such an agreement will work as a set of 'rules of the game' and is aimed at balancing flexibility and predictability, to keep the represented business process as focused, simple and manageable as possible.

1.3 Interaction protocols for integration

As mentioned in the previous section, for a set of participants to perform a business-process, some kind of agreement has to be met, in order to be able to coordinate and achieve the goals each participant is looking for. This is where the concept of business-protocol appears as an option for specifying an agreement and the set of rules that will guide the interaction.

Currently, the usual practice in order to create a business process among autonomous participants, is that each participant implements an interface in such a way that matches the interfaces of the other participants. (Weerawarana et al. 2005) shows how business processes are traditionally developed in a Web Service architecture. Development is done bottom up, in the sense that each participant is first described and designed such it will correspond to the others. Services are first described using WSDL and then, using some other WS-specifications, they are combined. This means that each part of a single business process is defined separately and from a different perspective.

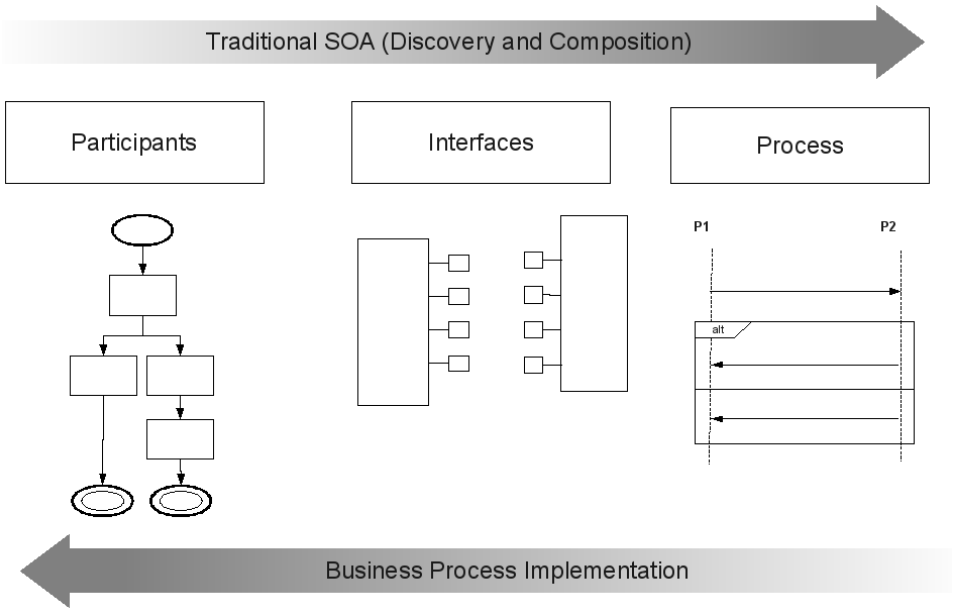


Figure 1.1: Business Process implementation cycle in opposite to Service composition in traditional SOA

This is natural for Web Services architectures, since these are based on the SOA concept discovering and invoking existing services. When a service is desired, one would look for a matching service that fits well in one's concept.

This approach does not go well with other situations in which the process exists conceptually before the actual participants are created. Figure 1.1 illustrates this: discovery and composition work by first having some services which define and publish their interfaces which are discovered and put together as a process in form of a composition. The situation being approached here is in principle in the opposite direction as it has been done with Web Services up to now. The present work intends to define a process first and then derive the the interfaces in form of Web Services and only then produce service implementations (the participants).

It is better to specify first what is the intention and what is to be accomplished and based on this clear definition create the tools or services that achieve this. It is very common to face situations where partners decide to perform together a specific business process. In this case, a centralized global view of the process is more desirable since issues about the cooperation can be cleared at this stage. This global view can serve later both as a contract between the participants on how the process is to be performed and also as the input for other participant end points or services.

Interaction protocols are defined precisely at this stage and serve the purpose of coordinating participants and specifying the cooperation from a

global point of view (Miller & Mcginnis 2008). Participants roles can be either adapted to comply with the protocol or created in order to fit them. Interaction protocols as a business model provide a good balance between a global centralized conceptual process, and a distributed and autonomous realization.

1.4 Contribution of multi-agent systems

The concept of complex interactions is very mature in the area of multi-agent systems, whereas traditional algorithmic approaches prove to be inadequate for one reason or another. Since processes are still specified as scripts, (for instance BPEL or OWL-S) they are limited, in essence, to constructs of normal procedural programming such as sequence, iterate, fork, and join which are not that different to job control languages (JCLs) of the mainframes of the 1950s (Singh, Chopra, Desai & Mallya 2004). Imperative instruction sequencing are more appropriate for implementing components (*the small*), interaction protocols are more suitable for putting independent distributed components together(the *large*) (DeRemer & Kron 1975). Procedural structures do not represent well the nature of complex interactions. FIPA (FIPA 2002) is an example of a proposed solution for modeling complex interactions. It proposes a stack which on top of it has a library of 'standard' interaction protocols which can be recombined and personalized to create models for business process.

Interaction protocols are a favorable technique to implement distributed business processes using Web Services, because they provide more freedom of (Singh et al. 2004):

- design: support heterogeneity, many different services with different supporting technology can be coordinated to work together.
- configuration: provide dynamism in order to support better adaptation and accelerate the evolution of a distributed system.
- action: provide autonomy of action for each participant. When the rules of the interaction are clear, each participant can decide autonomously which possible option to take in each turn.

The use of multi-agent techniques for automated negotiations is even more natural, as content rich communication is crucial. Participants must have a level of adequacy that can take advantage of the information being exchanged. Automated negotiation is a natural field of research in multi-agent systems. Agent oriented systems (Shin & Lee 2004) are capable of working with, producing and processing this content rich communication.

1.5 Summary

Distributed computing is experiencing a move towards more agile and flexible integration of systems. They are changing from simply having to pass data from one place to another, to represent or perform a Business Process. These business processes are increasingly involving business partners that are and intend to remain autonomous. Web Services and Service Oriented Architectures are the preferred tools to implement business processes. They are not used for service discovery and composition only, but also as a technology to implement processes that are established as a result of new partnerships. This is a development process that goes in the opposite direction as service composition. Instead of creating participants, looking for matching ones and producing as a result a distributed process, the whole process is first defined. This process serves as contract between participants and based on the information contained in it, the participating components are created. Interaction protocols are a very suitable solution for this purpose. Multi-agent systems technologies can contribute enormously in this area, since complex interactions and automated negotiations have been studied vastly in this area. But, in order to transfer techniques from multi-agents research to Web Services and SOAs, a low level integration of both scenarios has to be attained first.

Chapter 2

Objectives

Currently the most accepted technology for interoperability between heterogeneous systems are Web Services, but they have only been used to realize simple processes between organizations. Some mechanisms for enabling complex processes in the form of *orchestration* have been provided by industry, one of them is the Business Process Execution Language (BPEL)(Andrews & et. al. 2003). This work provides an approach for realizing complex interactions in a different manner, such that there is no dependency between participants, apart from the intrinsic dependency that the other participants fulfill their roles. Our approach is based on the principle of respecting the *autonomy* of participants, by not imposing an active ruling entity, an orchestrator that dictates the behaviour of the participants. Instead business processes will be modeled as conversations in a multi-agent system.

Using multi-agent technology, a framework for producing Web Services *choreography* will be proposed. Various aspects of the different stages in the development of such a system will be taken into account, from design problems to concrete implementation issues.

Our framework has a lower and a higher level. The lower level deals with concrete communication problems and grounding: using tools for Web Services, communication is implemented as for agents. The higher level deals with conceptual problems, like the design of complex conversations.

More concretely we shall use FIPA protocol specifications, since these are the most comprehensive and adopted communication mechanisms used in the multi-agent community. Our communication framework provides a basis such that agents (FIPA agents) and Web Services can transparently communicate. Messaging of a FIPA platform with Web Services will be proposed by grounding the FIPA abstract specifications using Web Services standards. This is done focusing on messaging only, other relevant issues like description and discovery are treated as a special type of contents to be transported using the proposed messaging.

At the higher level more complex tasks have to be solved. Taking advan-

tage of the vast amount of contributions in the area of interaction protocols and complex conversation between agents, we define a concrete model of protocol. This model includes all aspects necessary to produce business process models, moreover, the proposed model also provides the possibility of using engineering techniques like *classification* and *modularization* for coping with the complexity.

It is also the intention to produce concrete implementations automatically in the lower level following guidelines at the higher level. The possibility of working directly with abstract concepts as used in the higher level in a concrete lower level, like for instance, agents reasoning over protocol models and performing them in *run-time*, is also of interest for the present research, but not mandatory, since the objective is to produce a business process realization and not a new enhanced agent capable of planing its interactions. Even so, the proposed model is intended to suit this development area in the multi-agent community also.

2.1 Summary

The objectives are:

- To study Web Service communication
- To study agent communication, mainly the FIPA (FIPA 2002) agent platforms.
- To integrate the communication of FIPA agents and conventional Web Services in such a way that complex conversations can be implemented using Web Services.
- To produce a consolidating interaction protocol model that is comprehensive enough to describe the conversation completely. These descriptions have to be modular and reusable.
- To produce a implementation based on interaction protocol models. This implementation uses the integrating solution for agents and Web Services.

Chapter 3

State of the art

A fundamental component of the present research work is the study of agent-based communication mechanisms in a Service-Oriented Architecture. This chapter will cover primarily contributions found in literature about this topic. Contributions in different areas will be discussed to produce a description of the state of the art in Web Services and in multi-agent communication. Communication mechanisms in both areas will be characterized to establish a comparison. Also contributions in the area of interaction protocol modelling will be reviewed. This overview will serve as foundation for a modeling framework for business processes.

3.1 Web Services Standards overview

Distributed computer systems were already common in the early stages of computer science. In the seventies and eighties computer networks started to be frequent within organizations and in the nineties, the arrival of Internet made electronic communication between organizations very feasible. Even so, some other barriers were blocking integration, mostly differences between the diverse software products and the lack of standards for data transfer between systems. After a few years XML gained dominance and standards based on it started to show advantages over other mechanisms for data representation. It is in this period where Web Services were established.

Web Services are nowadays the preferred way for implementing Service-

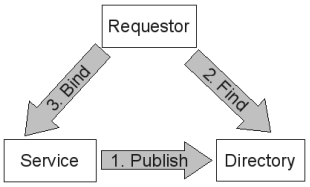


Figure 3.1: The SOA Triangle

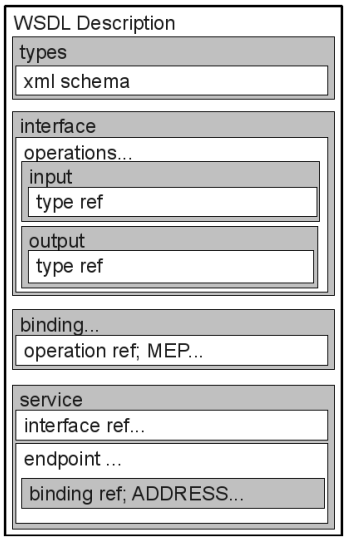


Figure 3.2: Structure of WSDL 2.0

Oriented Architectures. These consist of software entities, called services that can be searched for in directories, where descriptions of these services are published as shown in Figure 3.1(Weerawarana et al. 2005). Once discovered, services can be invoked to make use of them. Web Services standards consist of XML-based specifications for the structure of messages and the description of services. The World Wide Web Consortium (W3C) (W3C n.d.) is the organization in charge of the specification of Web Services. The standards provided by W3C relevant for the topic in discussion will be presented next.

3.1.1 Service description

Web Services Description Language (WSDL) (W3C 2002) is a specification for the description of services in terms of the messages that they interchange and their structure. WSDL descriptions are composed, as shown in Figure 3.2, of a specification of the types (data structures) to be used inside messages, the messages that contain them, the operations (concrete actions) that will make use of messages, and *port types* or *interfaces* which are collections of operations that are related to a specific task. Those components are part of the structural part of the description. WSDL also provides a mechanism for specifying a binding for interfaces, showing how interfaces are grounded, for instance using SOAP (W3C 2003). Finally, it is possible to create collections of these bound interfaces as a concrete service and associate a location for the service to be reached.

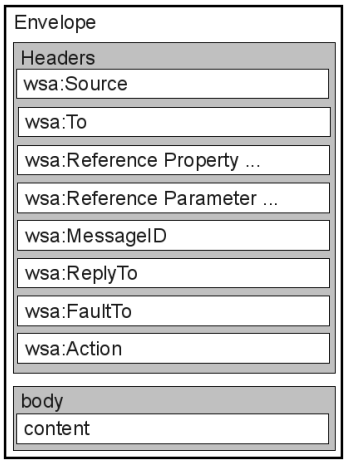


Figure 3.3: Structure of SOAP with WS-Addressing headers

3.1.2 Communication between services

Web Services also provide standards for messaging between systems. The message representation mostly used in Web Services is SOAP. On top of it, a new standard was provided for a more precise definition of messaging called WS-Addressing, both specifications are explained next.

SOAP

Simple Object Access Protocol (SOAP)(W3C 2003) is a specification to send messages using XML. It defines the structure of a message, organizes it in two main parts as shown in Figure 3.3, the headers block and the message body. The first one is used for annotating the message with information relevant for messaging and processing. The body is used for containing the information that is to be transferred by the message.

Addressing

Web Service Addressing (WS-Addressing) (W3C 2006*a*) is a specification of concrete headers used for message addressing. SOAP provides messaging using headers and WS-Addressing specifies fundamental headers and their semantics:

- `wsa:Source`: the sender of the message
- `wsa:To`: the receiver of the message
- `wsa:Reference Properties`: values that help together with the Endpoint Address to deliver the message.

- `wsa:Reference Parameters`: values that are not necessary to reach the endpoint, but to to interact with it.
- `wsa:MessageID`: an identification for the message
- `wsa:ReplyTo`: endpoint where to reply to
- `wsa:FaultTo`: endpoint where to send fault messages to
- `wsa:Action`: an identifier of the message's intent

WS-Addressing defines Endpoint References(EPRs) in order to address services. EPRs encapsulate the information necessary for reaching a service endpoint at run-time and are used, for instance, as data structure for some messaging headers like `wsa:Source, wsa:To,` etc (Weerawarana et al. 2005). An EPR can be extended with application specific parameters using `wsa:Reference Properties` and `wsa:Reference Parameters`.

3.2 FIPA platform and agent communication

This section will review how agent communication is modeled in typical multi-agent systems. The most significant contribution in this area are the specifications provided by FIPA(FIPA 2002). They have proposed one of the most widely accepted standards for agent platforms. This platform model will be taken as reference and representative standard of a multi-agent system platform, since it has been used as a base for other more detailed contributions that will be discussed later in this chapter.

3.2.1 FIPA Agent Architecture

The core of FIPA specifications is a model for agent platforms called Abstract Agent Platform Architecture (FIPA 2002*a*). It specifies how the platform is organized and how agents are identified. FIPA specified an abstract architecture for Agent Platforms independent of realization tools and that makes interoperability between concrete platforms possible while empowering solution-specific features. Therefore FIPA is focused on end-to-end support of a common agent communication language.

This architecture is a definition of how agents can locate and communicate with each other by exchanging messages. The architecture specifies an agent communication language (ACL) and certain components of the required infrastructure. These components are services available for agents and are in principle the following:

- **Agent Management Service (AMS):** (also Agent Management System) Is the service that manages agents in the platform and their life cycle.

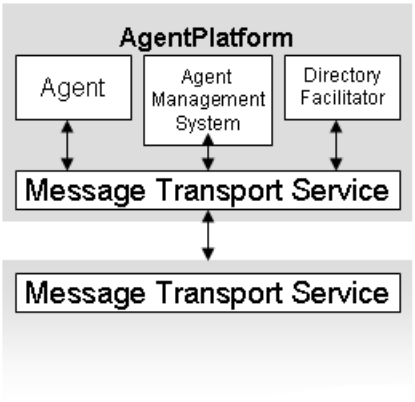


Figure 3.4: FIPA Agent Management Reference Model

- **Message Transport Service (MTS):** services that provide mechanisms for communication between agents in the same or remote agent platforms.
- **Agent Directory services:** a repository service for publication and discovery of agents. In opposite to AMS, this repository is properties oriented and not identification oriented.
- **Service directory services:** service for publication and discovery of services. Works similar to agent directories, but instead of agents, it manages services.

These conceptual components are reified in the Agent Management Specification (FIPA 2004). Figure 3.4 shows the structural organization of these components. Each agent platform complying to FIPA specifications will provide the 3 basic services: agent Management (AMS), Directory Facilitator (DF) which serves as agent directory and services directory, and Message Transport (MTS). Any agent entering an Agent Platform has to report to the AMS. After that, the agent can use the directories available for discovering other services and agents to interact with. These interactions are done using the message transport service.

One important aspect of this management model is the separation of the identification of an agent and the mechanism to reach it (its address). An Agent Identification (AID), as defined by FIPA, can be used to identify the agent in an absolute manner: the agent can change its location (its address) but keep its identification, which makes it possible for other agents to recognize it again later. There is in FIPA a strict separation between transport mechanisms (address) and identification.

3.2.2 FIPA Agent Communication

FIPA provides a stack of specifications for communication between agents. Figure 3.5 shows how communication specifications in FIPA are built one on top of another.

Interaction protocols (FIPA 2002*g*)
Speech Acts (FIPA 2002*e*)
FIPA ACL Message
Transport Message
Message Contents Representation (FIPA 2002*c*)
Message Envelope Representation

Figure 3.5: FIPA Communication Specifications Stack

Looking at it bottom up, the representation of the message envelope is the base. It contains transport specific information required in every single message, independent of their context, like: sender, receiver, how the message is encoded, etc. A way of representing the contents is specified on top of it. Envelope and contents can be realized on any appropriate representation technique, like bit efficient, Strings following a specific syntax, XML, etc. Message contents and envelope form together a Transport message which is the realization of a FIPA ACL Message (FIPA 2002*d*). On top of it, a set of ACL message types, called speech acts, define the different kinds of messages that agents will use. A clear semantic definition in correspondence to speech act theory (Sadek 1991) is provided for each speech act. There are certain well known situations in which a specific sequence of speech acts are expected. These sequences are called protocols and they are a pre-agreed and fixed ways of how messages are exchanged to serve a specific purpose.

FIPA Message Envelope

The envelope is an information structure that contains some fields for message addressing, correlation, semantic representation and context information. The structure is shown in Figure 3.6 (FIPA 2002*c*). It is composed of the following fields: sender, receiver and reply-to which are Agent IDs (AID) fields representing basic addressing information. An AID is composed of 3 fields, the name of the agent which is the unique identifier of the agent, an optional list of addresses where it can be contacted and an optional list of locators or directories, where the agent can be found. Next, a field called reply-by contains a time constraint for the response. After that, the envelope provides the message correlation fields: conversation-id, reply-with and in-reply-to. Reply-with contains the information token to use in the field in-reply-to of a response. Following that are the fields that describe the basic semantic of the message: communicative-act which contains the

Sender (AID)	Receiver (AID)
Reply-To (AID)	Reply-By
Conversation-ID	Reply-With
In-Reply-To	
Communicative-act	Protocol
Encoding	Language
Ontology	
User-Defined	
Content	

AID:
name
address
locator

Figure 3.6: FIPA Message Envelope and Agent ID (AID)

identification of the type speech act (from the FIPA speech act library) of the message and protocol which tells what protocol is being performed. Following that are fields that inform about details of the message content: encoding, language and ontology. The specification provides the option of user-defined fields for extension. Finally there is the field where the actual content of the message is placed.

Speech act

FIPA(FIPA 2002*e*) has specified the structure and semantics of a basic set of speech acts. Each speech act is an information structure associated to a specific semantic, like query, assertion, request, etc. They are defined in terms of preconditions and consequences that they have: Speech Acts are defined in terms of preconditions that have to hold for the action to be performed and the changes that the action produces, everything from the *perspective of the sender*. For instance, the `inform` speech act, the action of informing that a proposition p is true, is defined by the preconditions that the sender must hold that a proposition p is true, is uncertain whether the receiver knows the true value of p and intends that the receiver comes to believe that the proposition p is true. The complete set of FIPA speech acts can be found in (FIPA 2002*e*).

Interaction Protocols

In a multi-agents system, messages are expected to be exchanged between agents within a context and as part of an interaction. For this reason, FIPA provides a set of interaction protocols which are specifications of message exchange sequences that serve a certain purpose.

Interaction protocols are a mechanism that describes the different options of message exchanges possible in each stage of a conversation between agents. An interaction protocol allows different sequences of messages, the sequence that the conversation takes can be different from one situation to

another. In spite of that, they all belong to the same protocol, since a protocol works like rules of a game, but how each game looks like will vary from one case to another.

FIPA provides a set of protocol descriptions (FIPA 2002*g*) that describe from simple conversations, like an agent requesting another agent to perform an action, to complex interactions between several participants like auctions or brokering.

Interaction protocols are described using a variation of UML (Unified Modelling Language) sequence diagrams (Odell, van Dyke Parunak & Bauer 2000) which are graphical representations of the protocol. The semantics of these diagrams is specified in Agent-UML (Bauer, Müller & Odell 2000), an extension of UML with enhancements for agent specific issues like multi-threaded lifelines, roles and message semantics. This representation will be explained in section 3.5.6.

3.2.3 Work based on FIPA specifications

FIPA specifications served as base for several implementations in multi-agent systems. This section will describe two major developments that are relevant to our research: Jade and Agentcities.

Java Agent Development Environment (JADE)

Jade (JADE 2001) is an implementation of FIPA specifications in the form of a multi-agent system middle-ware which also serves as agent platform and provides basic services like directories and messaging. Its framework supports the implementation of ontologies for message contents and agent knowledge. It also provides FIPA's representation specifications like FIPA-SL (FIPA 2002*h*) in the form of "String representation", a syntax very similar to LISP.

Jade is also one of the preferred platforms to implement complex conversations between autonomous agents, because it provides a library of behaviours for performing FIPA interaction protocols. New complex conversations and their corresponding behaviours can be produced from scratch or by combining preexisting behaviors. Therefore is Jade a very suitable tool for implementing business processes using multi-agent systems.

Agentcities

Agentcities was the first world wide open network for agents(Agentcities 2002) that took advantage of the standardization that could be achieved with FIPA Specifications. Agentcities' network is composed of different platforms implementations, all based on FIPA standards. The network hosted a wide variety of projects.

One relevant achievements of this project was to serve as laboratory for global integration of heterogeneous agents and also to show the potential of such a big integrated community. It worked for various years and already has many recommendations based on the gained experience. Concerning integration of Agents to Web Services, Agentcities recommended a model (Agentcities Web Services Working Group 2002) whose main contribution are two components, called gateways, for mapping interactions from the perspective of the AP to Web Services and vice versa. Requests initiated by agents are translated to the Web Services representation and the response is translated back to ACL. The same is done correspondingly for interactions started by a Web Service consumer to a service provided by an agent in the AP.

This approach follows a recommendation for integration of non-Web Services or legacy platforms into Web Services: any system using technologies other than Web Services is to be integrated to Web Services by using an *adapter* (Barry 2003). Adapters are converters or translators between the local technology and Web Services.

3.3 Comparison of Agents and Web Services Specifications

The properties described in the previous sections enables us to make a comparison of both specifications. This comparison, shown in Figure 3.7, will serve later as a basis for our proposed integrated architecture.

The first aspect to compare is low level groundings, present in both. The goal of Web Services is to have a specification that allows communication between systems using XML. Web Services are normally grounded using SOAP for message structure and XML for contents. It has well established *functional* description languages that allow detailed descriptions of services and collaborations between them. FIPA provides a set of specifications that go from the basic grounding to the abstract layers of communication like semantics and complex interactions. It has grounding specifications in different formats including one for XML, for the envelope as well as for the contents. The architecture allows the addition of new groundings. FIPA specifications go further up in the abstraction level and provide specifications for semantic and structure of high level concepts left completely open by Web Services, which focus on bare messaging.

Web Services are widely adopted among a vast amount of applications, their main interest is interoperability in any application area. FIPA specifications on the other hand, focus on interoperability and specification of agents and therefore have moderate to good adoption inside the multi-agents community, but are practically nonexistent outside of it. The wide adoption of Web Services is the most attractive reason for producing an integrating

Aspect	Web Services	FIPA
Syntax and Structural definition (Grounding)	Detailed	Detailed
Adoption	Vast	Little
Semantic definition	Little, open room for application semantics	Well established, open room for application semantics
Communication stability support	Explicit	Simple and not always explicit
Communication complexity	RPC only moving towards complex interactions	Targeted at complex dialogs
Complex Conversations support	Immature	Mature
Reasoning capability of the participants	Reduced	Complex
Identification	Address	Instance ID
Statefulness	Stateless	Stateful

Figure 3.7: Web Services and FIPA Specifications comparison

architecture.

FIPA specifications have given less importance to some key concepts for low level communication, like acknowledgement of messages or detailed exception handling description. In FIPA sending a message and not receiving an error is intrinsically perceived as a successful transfer of the message, which is not always the case in conventional communication systems. In Web Service specifications, exceptions for instance, are part of the basic vocabulary for the description of choreographies (W3C 2005). Even though FIPA provides speech-acts and protocols for handling exceptional situations, these are left relatively lax and not targeted to be as explicit and detailed as in Web Services.

Communication is very complex in multi-agent systems and one of the main subjects studied in this area. Web Services, on the other hand, have been traditionally simple, as far as conversation patterns is concerned. Most of the times only simple request-response patterns are used. Even though Web Services are suitable for more complex communication patterns, deployment of complex communication systems is still not wide propagated in this area. Usage of Web Services in complex interaction processes is still in an immature stage compared to multi-agents systems. Only orchestration tools like WS-BPEL(Andrews & et. al. 2003) have shown success. FIPA

specifications target autonomous agents expected to have enough reasoning power to manage complex conversations, in opposite to Web Services, where the objective is to provide interoperability between heterogeneous systems which not necessarily support complex reasoning capabilities concerning communication.

Web Services, in current practice, are focused on simple RPC functionality. Participants are traditionally simple (to the outside) and messages can carry complex information, but their interaction-related semantic stays simple. FIPA is focused on complex interactions with messages having simple to complex contents and semantics and participants that can be simple but also complex reasoning agents.

Finally, another aspect relevant to compare is the statefulness of the agents participating in conversations. Agents are in essence stateful and FIPA specifications treat them like that. Agent identification is part of the lower communication layers. Web Services focus on addresses, the actual agent providing the service is irrelevant. Web Services focus on messaging only and treat services as stateless instances. This is clearly reflected in the way Web Services and FIPA specifications identify participants, namely using Endpoint References (addresses) (W3C 2006*b*) and agent instance names (FIPA 2004) respectively. Even so, Web Services are slowly starting to support stateful participants and part of this effort is the specification of WS-Addressing.

3.4 Integration of Web Services and multi-agent technology

The multi-agent community was for years interested in integrating multi-agent systems and Web Services. This section will provide a brief review of the most relevant proposals and implementations provided during this period. Their solution will be briefly explained along with some relevant aspects and background. This section serves also as a survey of Web Services and Agents integration solutions and will be used later to compare against our solution.

3.4.1 Agentcities Recommendation

Agentcities provided experience in global scale agent integration using FIPA standards and specifications. As already mentioned in Section 3.2.3, it recommended an approach for Agents and Web Services integration. This was one of the first recommendations about this subject and was based mainly in the general procedure used to integrate non-Web Services systems with Web Services (Barry 2003). It consists of the addition of an *adapter* layer between the system to be integrated and the Web Services platform (Agentcities Web

Services Working Group 2002). The purpose of this adapter is to map Web Services concepts to Agent concepts and vice-versa. This recommendation was followed by most of the implementations that will be reviewed next. They all are based around the concept of a *gateway*, which works as an adapter.

3.4.2 Integration Agent Gateway

Taking the recommendation proposed by Agentcities, Whitestein technologies (Greenwood & Calisti 2004) has proposed a Web Service Integration Gateway Service (WSIGS). This gateway is present inside an agent platform as an agent service published in a directory facilitator. The WSIGS provides certain services like: publishing and maintaining Web Service descriptions in the internal DF and executing request to the published services. It is also a Web Service published in a UDDI for Web Service consumers to discover and use services offered by agents inside the platform. Requests coming from one end to the other are intercepted by the gateway and the contents is converted to the format corresponding to the target.

This approach works on a dual descriptions storage, each description is present as WSDL and service description (SD) in the UDDI and the Directory Facilitator(DF) of the Gateway respectively. A mapping between them and translation modules use these dual descriptions to convert messages form one encoding to the other. For this purpose, the WSIGS needs to keep track of all services offered by the agents, as well as all UDDI entries of services that could be called by an agent.

The WSIGS is composed basically of three main components:

- Gateway Registry: this part of the WSIGS is in charge of providing the directory related services. It contains an UDDI registry, an internal DF and modules for translating messages and descriptions from ACL to SOAP/WSDL and vice-versa. Requests for publication or discovery are translated and attended by the corresponding service, for Web Services requests the DF and for Agent requests the UDDI registry.

- Web Service invocation component: it intercepts ACL encoded requests coming from agents and produces a corresponding service description (SD). This SD is then used to search in the internal DF for the service required. Once found, the corresponding WSDL in the internal UDDI is used to generate the SOAP request to the targeted service.

- Agent invocation component: this component works in the opposite direction of the Web Service invocation component. It is important to remark that each agent service is represented as a stub inside the WSIGS. Requests by Web Service clients are made to these stubs which

get the corresponding SD from the internal DF to generate then an ACL message to be sent to the agent.

If the corpus of services to be used by agents is not too big, this approach can be a practical solution for enabling access of agents to Web Services and also offering their services as Web Services. As mentioned in (Greenwood & Calisti 2004), this approach does not support more complex conversational patterns. It only supports the standard Web Services request-response pattern.

3.4.3 FIPA TC Services

The initiatives mentioned above motivated FIPA enough to take actions to integrate FIPA agents and Web Services (Dale & Lyell 2003). The need of such an integration was clear and therefore, some of the latest Technical Committees (TC) were approaching this issue. Among them, the most relevant was the Services Technical Committee (TC Services)(2003). It focused on providing a service meta-model to ground common service models currently used, like Web Services.

The unstructured notion of a services in FIPA Agent Management specification was one of the main problems. The specification does not approach service composition. TC Services would work on a specification of an Abstract Service Architecture that is more compatible with current standards. More details about the architecture would be taken into account, such as different description levels: semantic, functional, models, etc. This group believed that FIPA technologies could contribute in the further development of Web Services towards *advanced services* that implement complex dynamic business processes. TC Services explored concepts like Service Orchestrator and Composer Agents for extending the Abstract Service Architecture. This group stopped when FIPA suspended its work around 2004. For that reason no concrete specification or implementation of any of these concepts was accomplished.

3.4.4 WS2Jade

WS2JADE (Nguyen & Kowalczyk 2005) is a tool that produces agent impersonators of Web Services that serve as adapter for the service in a Jade Platform. It has the advantage of being able to integrate such a services at run-time. This makes discovery and invocation of Web Services during run-time possible. This approach is also capable of implementing complex conversations, since all services are represented inside the agent platform by proxy-agents and these proxy-agents can participate in any conversation as any other agent in the platform. This tool is not intended to provide agent-services in the platform as Web Services to the outside. They even remark the lack of theoretical work on the topic of agent to Web Services

integration, specially due to the stateful asynchronous communication of agents against the stateless and dominantly synchronous communication of Web Services.

This tool is also using replication of the Web Services world inside the agent platform. It is therefore not suitable for situations in which there will be interaction with a big amount of Web Services.

3.4.5 AgentWeb Gateway

This gateway(Shafiq, Ali, Ahmad & Suguri 2005) is composed of 3 bi-directional translators:

- Search query converter: converts registry queries to let agents look up in UDDI registries and Web Services in DFs.
- Service Description converter: converts description publications for agents to publish their services in UDDI registries and Web Services in DFs.
- Communication protocol converter: converts service invocations in both directions.

AgentWeb Gateway allows communication and usage of infrastructure in both directions. Both of the environments are left untouched and keeps duplicate functionalities, specially in the case of publication and discovery.

3.4.6 FIPA Agents and Web Services Integration (AWSI) working group

In 2005 FIPA started again as a sub-organization of IEEE. As part of this restart, a group called Agents and Web Services Integration (AWSI) was created. This group presented its proposal(Greenwood, Lyell, Mallya & Suguri 2007) which reviews the different contributions and provides a general view of how to achieve integration better. Some of these are gateways or adapters that have already been discussed in the present review. This proposal includes our proposed architecture, presented in chapter 4. Further details will be discussed as results in Chapter 8.1.10. AWSI also takes into account solutions in the area of complex conversations. This part of their contribution will be reviewed in the next section.

3.5 Complex Interactions

Complex conversations, widely studied in the multi-agent community, have attracted the attention of the Web Services community. This section reviews the state of the art of important contributions in the area of complex

conversations, business process modeling and interaction protocols. First, some standards and techniques used to produce business processes will be reviewed followed by contributions from the multi-agent community.

3.5.1 RosettaNet

RosettaNet (RossettaNet 2002) is a consortium committed to the creation and implementation of XML-based, open standards for electronic business. The intention is to form a common e-business language for supply chain partners to coordinate. This language is used to define Partner Interface Processes (PIP) in terms of a transaction, the vocabulary and the business process. PIPs are either a notification or a request-response pattern (Damodaran 2004). There is a set of predefined PIPs for common situations like purchase management, manufacturing and inventory management, they are distributed in seven segments (Jilovec 2004):

1. Partner and service profiling and review
2. Product information
3. Order management
4. Inventory management
5. Support and service
6. Design exchange, configuration and quality assurance

The objective of RosettaNet matches in some extent our work, because it automates the public process between trading partners. Integration is approached by establishing documents and exchange standards. In this approach, participants are autonomous, not centralized entities performing the processes.

The most important contribution of RosettaNet is the provision of standardized document descriptions to be used in XML-based messages. This specification is an improvement over old and widely disseminated practices like EDI (Kantor & Burrows 1993). These documents are very detailed, which is suitable for automating processes. It also organizes interactions in areas and in a way that makes it easier to combine them. PIPs are simple, they can be rather limited for more complex processes and can also result in rigid processes. PIP is a solution for aligning information systems belonging to different organizations and not a tool that reflects a business process.

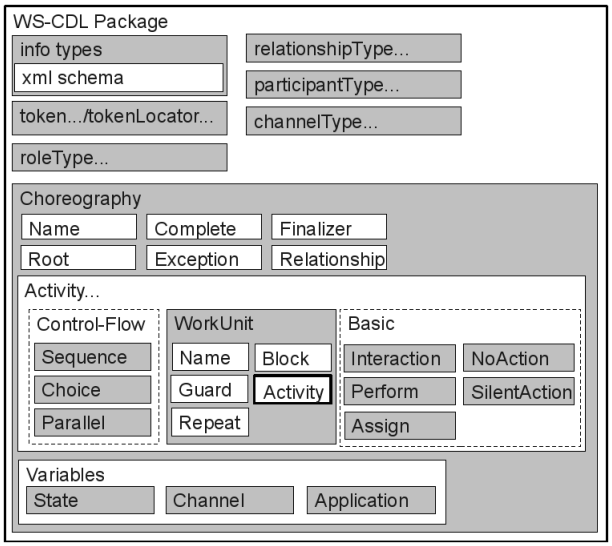


Figure 3.8: Structure of Web Services Choreography Description Language

3.5.2 WS-CDL

Web Services Choreography Description Language (WS-CDL) is an XML language developed by W3C to describe from a global point of view peer-to-peer collaborations (W3C 2005) between Web Services. It describes processes that involve different participants, in terms of messages that are exchanged. It is based on WSDL descriptions to represent the endpoints and XML types to represent messages. It provides also mechanisms for describing choreographies independently of the actual services that will participate, replacing them with the concept of role. These roles are represented by WSDL interface descriptions. Similarly, the concept of token is introduced to make WS-CDL descriptions independent of actual types used in messages. It provides convenient resources for making collaborations reliable, like process blocks for exception handling, finalizing processes and synchronization between participants.

A description of a choreography in WS-CDL is an XML document containing a WS-CDL package whose structure is depicted in Figure 3.8 (Barros, Dumas & Oaks 2005). First, it declares some types for:

- Information: these contain plain XML-Schema descriptions of information types that can be used as content of messages exchanged in the choreography.

- Tokens: aliases for pieces of data inside variables of messages that are significant or are needed in the description of a choreography. Tokens provide an intermediate layer to decouple the choreography description from concrete XML types. They serve as an abstraction of a concept

relevant for the description that can be represented syntactically with any XML type. *Token-locators* are XPath expressions used to link a token in a WS-CDL description to a type in an XML document. By changing solely the token-locators, the same choreography can be used in different scenarios with different documents.

- Roles: is an interface composed of a set of *behaviors* a participant can exhibit. Behaviors are observable connection points of a role.
- Relationships: associations established between 2 roles and the list of behaviours that are to be provided by the roles.
- Participants: participants are aggregations of roles that conform the description of a participant in the choreography.
- Channels: describes how and where information is exchanged between participants. These can be understood as connection end points provided by a participant. Using the information in a channel type, information can be provided to the participant. ChannelTypes descriptions specify how a channel can be used: for a single, for distinct or for shared interactions.

The main part of a WS-CDL document is called a *choreography*. It defines rules for the exchange of messages. In every packages there will be always a maximum of one top-level choreography. A choreography has some attributes that serve to describe its nature:

- Name: identification of the choreography
- Root: to mark the top-level choreography
- Complete: to explicitly complete a choreography. It describes how to evaluate if the choreography has been completed using a boolean XPath expression.
- Exception: a *Workunit* to handle cases of exception
- Finalizers: to specify finalizing *Activities* of the choreography
- Relationship: the relationships the choreography participates in

Work-units are used to describe constraints required to be fulfilled in order to advance in the choreography. They annotate *activities* with a *guard* which is a condition for the activity to be performed. If the guard evaluates to false, the activity cannot be performed. The attribute *block* defines if the work-unit has to block in case variables in the guard condition are not available or the guard condition is disabled. *Repeat* tells if a Workunit can be repeated after the guard has been evaluated to true.

A Work-unit has an *activity* which can be either a control-flow activity like a sequence, choice or a parallel, or it can be a basic activity like:

- Interaction: represents an information exchange between two participants over a channel.
- Perform: mechanism to embed a choreography inside another. This activity specifies when and how another choreography is performed.
- Assign: variable assignation. Creates or changes values in a Role.
- NoAction: marks where explicitly no action will be performed.
- SilentAction: used to mark when a participant performs non-visible activities

Finally a WS-CDL package has the definition of variables to be used in expressions inside the choreography. These variables can be state variables capturing information about changes at a RoleType, or channel variables or application variables containing information being exchanged.

WS-CDL has been the most comprehensive effort in W3C to define a specification for peer-to-peer complex interactions. Even so, after some years, its success has been very limited. This has many reasons (Barros et al. 2005), like poor linking with other WS-* standards, but there are mainly two big flows that stand out:

It lacks features to include cardinality of participant-types in a description. There are no means to describe situations like auctions, where a first part of the interaction is performed with multiple participants, all of the same type, a fundamental concept for choreographies. It is therefore not possible to define constraints based on participants or interactions cardinality. For instance, when an activity is to be performed by a participant only after it has received a certain amount of replies.

Another inconvenience is the nature of WS-CDL as a tool for description. The nature of a design task is very different than a programming or execution task. As one can see, WS-CDL inherits many characteristics of low level imperative programming, but does not provide a good solution for design, like a visual representation. WS-CDL , in opposite to tools like BPEL, is mainly intended for abstract, high-level design and not for execution. Therefore, just an XML language is not enough. This has been one of the main reasons why WS-CDL has not gained importance against other design tools like UML.

3.5.3 Business Process Model Notation (BPMN)

Business Process Model Notation (BPMN)(OMG 2009) is a diagrammatic notation for modeling business processes. Its objective is to provide portable and visual process descriptions that can be used in diverse tools and environments. It is intended for a diverse audience beyond just IT developers.

Processes in BPM can be of three kinds:

- Processes (Orchestrations)
  - Private non-executable: specific or internal of a participating organization intended for documentation mainly.
  - Private executable: specific or internal of a participating organization intended to be executed
  - Public: interactions between participants
- Choreographies: a definition of a contract that describes the expected behaviour of interacting participants. They are modeled similar to private processes and are composed of activities which involve 2 participants and that are not controlled by any centralized observer.
- Collaborations: can be processes or choreographies

BPMN provides a comprehensive set of graphical elements with specific semantic definition for diagrams that represent business process models. This is an alternative to other tools used for modeling processes like UML. It provides a more intuitive appearance suitable for a more inter disciplinary development. It fits well in a model driven architecture (MDA) as a computation independent model (CIM) that can be used as input to develop distributed systems. Its view covers from internal to observable aspects of the model.

A representative subset of the diagram constructs used in BPMN will be explained based on the example in Figure 3.9:

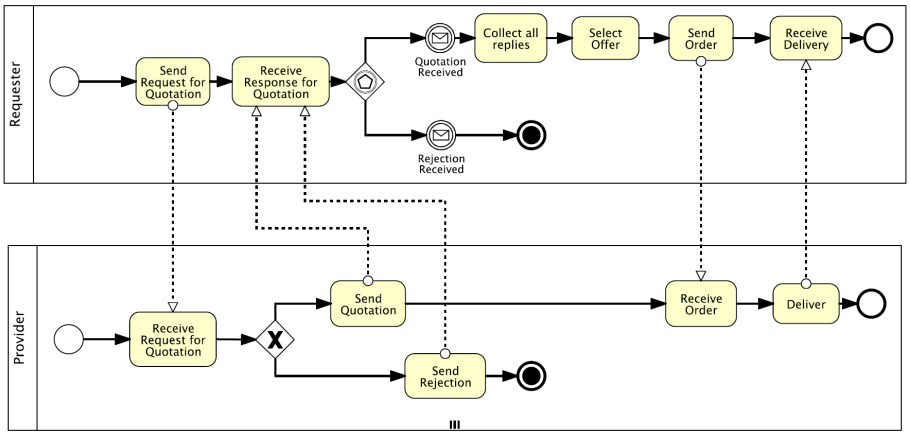


Figure 3.9: BPMN Example: Order Process

- Pool: represents a participant in a conversation. Used to contain the process belonging to a participant in a distributed business process. In the example diagram, there are two pools, one called Requester and one called Provider.

- Activity: represent specific tasks done in a process. Chains of activities make processes and it is also possible to use an activity to represent a sub-process. They are represented using rectangles with rounded corners, for instance: *"Send Request for Quotation"* or *"Deliver"*.

- Events: events that start, interrupt or end a process. These events can be of several kinds: Message, Timer, Rule, Multiple, Error, Compensation or Terminate loop. Events are circles and their special types are represented by the different icons they may contain. In the example there are starting events at the beginning of the process for each pool, two message events representing the two kind of messages that can happen for the Requester. There are also ending events for the normal branch of the processes and *termination* events for the cancelation branches. Termination events have a black circle inside them.

- Gateways: used to control how processes diverge and converge. They are represented using diamonds and also have different kinds, represented by different icons inside the diamond. Gateways can be data-based exclusive, event-based exclusive, both representing unique choices, inclusive that represent multiple choices, complex for advanced decisions and parallel where multiple parallel paths are defined. In our example there is one data-based exclusive gateway marked with a "X" in it, in the Provider pool. It represents the choice of sending a quotation or rejecting the collaboration. In the Requester side we have an event-based gateway with a pentagon in it that represent the two branches where the process split depending on the message event that happened. To define each of the options for the choice in the event driven gateway, message events follow each of the two options, defining one for the case when a *Rejection* is received and one for when a *Quotation* is received.

- Connectors: used to connect the different elements in a diagram. They can be *sequence* flows represented by solid arrows used to define the sequence of activities in a process. *Message* flows are dashed arrows with a circle where they start. These represent message exchanges between pools. These two connectors are used along the whole diagram. There is a third kind of connector, an *association* that is not present in th the example. Associations are used to connect data, information and artifact objects to flow objects.

- Data Objects: used to represent relevant data objects to elements in the process.

- Artifacts: these are extensions for defining other diagram objects.

3.5.4 WS-Business Process Execution Language (WS-BPEL)

Web Services Business Process Execution Language (WS-BPEL or simply BPEL) (Andrews & et. al. 2003) is an extensible XML workflow-based language for defining Web Services compositions that can perform (Weerawarana et al. 2005):

- Flexible integrations
- Recursive compositions
- Separation and compossibility of concerns
- Stateful conversations and life cycle management
- Recoverability

A BPEL process is a composition of activities of two kinds:

- Basic: in- or outbound Web Services interactions, specific data manipulation actions or actions like `empty`, `wait`, `terminate`, `throw` and `compensate`.
- Structured: contain other activities defining their logical relation. They can be `sequence`, `switch`, `flow`, which is a structure of parallel actions, and `while`

Abstract and executable processes are constructed using activities to describe the business process. Abstract processes are message exchange protocols that describe the externally visible interaction between participants without the internal processes running inside partners. Executable processes are the logic that is actually performed inside each partner. Singh (Desai, Mallya, Chopra & Singh 2005*b*) claims that BPEL is a mixture of these two layers, interaction and internal business logic. Even though it is a very well accepted tool for composing services, it does not solve our objectives, it is a language for hierarchical orchestration intended to be executed by a BPEL engine which plays the role of the orchestrator. Following the BPEL process, the engine invokes the services being composed and performs data manipulation. It has a very algorithmic structure for programming distributed processes the same way as program modules are done.

3.5.5 BPEL4Chor

An extension of BPEL for modeling choreographies has been proposed, taking advantage of the acceptance a standard like BPEL enjoys (Decker, Kopp, Leymann & Weske 2007). It is a layer used on top of existing BPEL concepts making it a language suitable also for representing choreographies.

BPEL4Chor consists of the the following three concepts:

- Participant Behavior Descriptions (PBDs): it comprises the communication activities and the control and data flow dependencies between them. It is a description of the sequence of actions a participant must perform in order to fit in the choreography. This description remains in an abstract level. For this purpose, statements in BPEL are used with the exception of some used to ground concepts to the concrete implementation, like `partnerLink, portType` and `operation`. PBDs in BPEL4Chor are not that different than any abstract partner description in BPEL. They are behaviour descriptions in a form very similar to structural programming.

- Participant Topology: this is where a choreography is mainly represented. At first the actual participants are declared using PBDs to define them. In the case where an unknown amount of participants of a same type is required, the concept of `participantSet` is introduced. The cardinality of these participants is explicitly defined with an attribute called `forEach` that associates the participant with the action that starts the conversation with it. Communicative actions between participants are represented using `messageLink` statements. It is composed of a sender reference, its sending action (`sendActivity`), a receiver reference, its receiving action (`receiveActivity`) and a `messageName` attribute representing the type of message. Using `message-Links` the participant's behaviors are interconnected to create the choreography. The sequence of actions in the conversations is therefore ruled by the PBDs.

- Participant Grounding: introduces the grounding details and mappings to Web Service specific configurations that were taken out in PBDs. The purpose of having this aspect separated, is to make models described with this language reusable.

One fundamental advantage over other choreography languages like WS-CDL is that it already supports parallel interaction with several partners which is a fundamental feature a language must have in order to model communication patterns in a SOA. Another advantage of this tool is that it can model combinations of BPEL orchestrations and choreographies in one single model.

They say, the PBD could be done after the topology, depending on the approach being bottom-up or top-down or that it could even be avoided if it is not of interest, but as far the specification in the publication goes, this is not possible, since PBDs define the sequencing and therefore the core of the conversation structure. The way rigid behaviors of participants are connected by simply establishing relationships between actions of different participants makes the approach to have poor modularity properties. They produced a browser-based visual representation based on an extension

of BPMN (Decker, Kopp, Leymann, Pfitzner & Weske 2008) along with a mapping from this model to BPEL4Chor. Having a visual tool significantly improves the ease of use for this tool for designing choreographies. Verification of certain properties like absence of deadlocks are done by producing a model of the choreography using Petri-Nets. Only diagrams that use block-structured constructs only or control links can be produced with this tool, patterns like multi merge of arbitrary cycles are not supported since they do match the process driven coding model of BPEL. It also inherits a major characteristic of BPEL and BPMN: mixing internal process models of the participants with the visible part of the choreography. As a matter of fact, the choreography structure is dictated by the internal process of the participants.

3.5.6 AgentUML

One of the most important techniques for modeling interaction protocols are swim lane UML sequence diagrams. This kind of diagram is very helpful, because it focuses on modeling directly the desired behaviour that emerges from the behaviour of the participants, without specifying a centralized execution control.

But, in principle, UML sequence diagrams have some difficulties when used for modelling interaction protocols. Sequence diagrams were originally created to describe interaction sequences between instances of a system, in other words, they represent concrete conversations in run-time. Instead, interaction protocols are intended to model how conversations can be performed and not how a specific one was performed. For that reason, a new model was created to cope with some of these aspects (Bauer et al. 2000).

AgentUML is a model that takes UML sequence diagrams and enhances them to represent protocols. It introduces some concepts like alternative behaviour and participants cardinality constraints. This way some of the basic problems are solved, like turn taking and valid message sequences modeling.

Figure 3.10 shows an example of the ContractNet protocol modeled using AgentUML. The ContractNet initiator sends a call for proposal to m participants. From these participants, n reply within a deadline, i do it with a $refuse$ and the rest (j) reply with a *propose*. From these j, k are rejected and l are accepted. Those accepted can reply with a $failure$, $inform-done$ or $inform-result$. Each lane is enriched with rectangles that extend along them. These rectangles represent the periods where the participant is active in the conversation. A participant can only perform sequences of actions that are connected by a single rectangle. For instance, when an initiator receives a *propose* it can reply with a $reject-$ or an $accept-proposal$ and then wait for replies from those accepted. On the contrary, when an initiator receives a $refuse$, there are no more actions to be performed and that

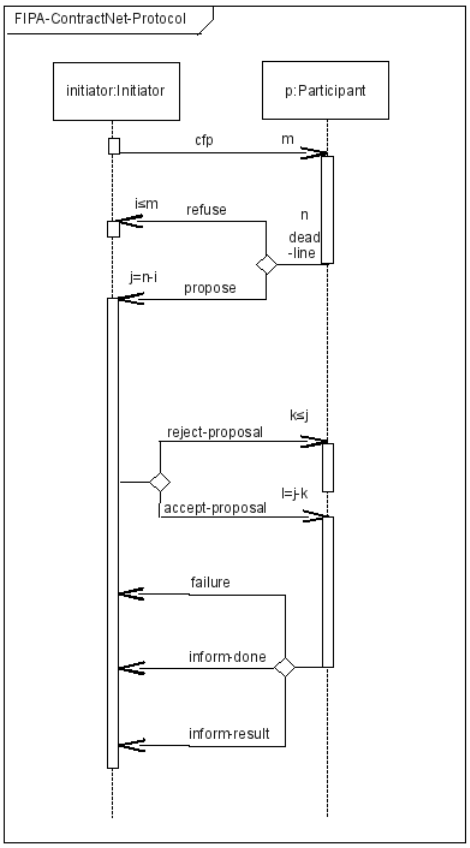


Figure 3.10: FIPA AgentUML protocol example

conversation is ended.

AgentUML has been the preferred tool for describing interaction protocols, since they provide intuitive diagrams capable of describing many aspects at the same time. Many contributions use them to illustrate protocols in their solutions, to describe new protocols or even to build on top of them new tools. Ehrler and Cranfield (Ehrler & Cranefield 2004) have developed an EMF model of AgentUML and implemented an automatic interpreter capable of performing the protocols, using a platform called PAUL (Profile for AUML Linking). It looks for incoming and outgoing message events and executes the code as specified by the diagram and some extra annotations that define how decisions are to be taken. In order to interpret the protocol, it is decomposed in descriptions for each role, based on events and code associated to it. An interface is produced for each role and its implementation is called for each event as specified by the protocol.

3.5.7 UML2 enhancements and AMP proposal

UML is a very suitable and widely accepted diagram specification that is very useful for describing interactions between objects. There are propositions on using UML2 for modelling collaboration between Web Services (Kramler, Kapsammer, Retschitzegger & Kappel 2006). Using several kinds of UML diagrams they model different aspects that go from basic information aspects, to interactions, services up to business and market models. But there are still conformance problems in the way UML sequence diagrams are used in agents and interaction protocols as proposed by AgentUML.

As part of an initiative to produce an UML modeling profile for agents (AMP) Haugen (Haugen & Runde 2009) proposes some necessary enhancements for protocol meta-models in order to be able to model them using UML 2. This enhancement is based on previous proposals like those in AgentUML of Section 3.5.6 which are not valid UML 2 diagrams. It recognizes that those proposals fail beyond the informal level, since they do not define important aspects in the conversation like which participants take part in which lifeline of an interaction protocol. They propose to model this by introducing a *Subset notation* on messages in an UML Sequence diagram as the one in Figure 3.11.

Sequence diagrams were used for describing interactions in a specific execution situation. Therefore their lifelines represented specific *instances* and not a type representing role, as intended by AgentUML diagrams. In a conversation between agents, as the one in the example, which is between an Initiator and many participants, conventional sequence diagram are not suitable, since it is desired to describe how a conversation is to be done with several participants. What is clear is that participants, in this example, will have only 3 possible ways of exchanging messages: they either propose or refuse the call. Those proposing will receive an acceptance or a rejection from the initiator. These possible paths in a conversation are used to define subsets of participants, in this example *refused, rejected* and *accepted*, for those who refuse to participate, are rejected or accepted by the initiator respectively. The advantage is that it now allows to represent the specific possible ways a conversation can take, by describing the conversation between the initiator and the subset of participants that take each specific way. These diagrams do not describe an interaction between specific instances, but between sets of them that share the same lifeline of conversation. Each of these sets conform a partition of the set of participants and multiplicity constraints can now be associated to them.

Instead of making a lifeline for each of the subsets, a special notation is added in the diagram. The **all** keyword followed by a *part name* is used to annotate messages to tell that a message is multi-casted to the set specified by the part name. For instance the message cfp is multi-casted to the complete set p of participants, because it is annotated with the statement

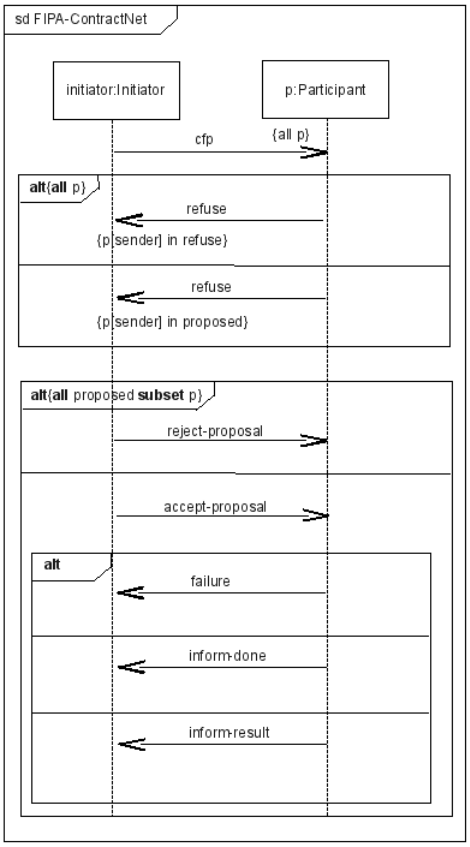


Figure 3.11: FIPA protocol with subset notation

{**all** p}. This notation is also used to define iterations of blocks. This means that such a block exist for each participant in the subset and that all blocks in the iteration are combined in an enclosing parallel construct. An example is the first **alt** block that is annotated with {**all** p}, meaning that for each participant in the set p, an alternative option is possible (for each participant in parallel), between replying with a refuse or a propose. Also the **alt**{**all** *proposed* **subsets** p} means that all participants in the subset *proposed* can receive alternatively either an accept or a reject.

The **all** construct is defined as follows:

$$\{\textbf{all } s \textbf{ subsets } p\}\ d \stackrel{def}{=} \textbf{par}_{p' \in s}\ d[p'/p] \tag{3.1}$$

where $d[p'/p]$ is the diagram d with the lifeline p substituted by the lifeline p' which is the lifeline of an element of the subset s, an instance of a participant member of the subset s.

On top of this, some other peculiarities of FIPA protocols are modelled using this technique. In FIPA protocols, all participants are allowed to send

a *not_understood* message at any point in a protocol to announce that a message was not understood. This is modeled with a simple protocol that is composed of a *not_understood* message within an **opt** (optional) block. This block runs in parallel (**par**) to any protocol. Similarly there is another peculiarity dealing with the possibility of the initiator to cancel a specific conversation any time. This can be modeled similarly as the *not_understood* case, but this time **opt** block surrounding the *cancel* protocol is annotated with {**all** p} meaning that this option is performed in parallel to all participants. As a result, the *cancel* protocol can be performed for each conversation without affecting any other conversation with other participants started in the same ContractNet.

A final component to the enhancements is the specification of time constraints. For instance, to enforce that $reject-$ or $accept-proposal$ are sent after a certain period d and that only proposals collected within that period are taken into account, some annotations are added. The time when the cfp is sent is annotated with $@t$ meaning that the message is sent at time t. Then the replies $refuse$ and *propose* are annotated with the *constraint* $\{t..t+d\}$ meaning that they must happen in the period between t and $t+d$. This same kind of constraint can be added to the messages $reject-$ and $accept-proposal$ to force them to be sent after all proposals are collected, using the following annotation: $\{t+d..\infty\}$.

One more detail to cover is that UML2 diagrams base their constructs on operations of the form `alt`, `seq`, `loop` and `par` which correspond to alternative, sequence, loop and parallel respectively. Using these constructs produces models that tend to be more algorithmic oriented.

3.5.8 Dialogue games

Dialogue games are a logic based formalism for modeling interactions between autonomous agents (McBurney & Parsons 2002), where the action of sending a message is modeled as a game move. Dialogue games have originally been used to study, describe and explain why participants in a dialogue perform specific speech acts or to define the rational process necessary to decide what action to take in a dialogue. This is an approach used in areas like linguistics and philosophy and has been introduced to the area of multi-agent systems to model interaction protocols (Amgoud, Maudet & Parsons 2000).

Dialogues are classified based on their purpose in persuasion, information seeking, negotiations and deliberations. They are defined by a set of rules organized in the following components:

1. Commencement Rules: definition of the conditions under which a dialogue starts.

2. Locutions: which speech acts are permitted.

3. Combination Rules: define which locutions are permitted in a specific context.

4. Commitments: when a participant establishes a commitment to a proposition based on the moves it makes.

5. Termination rules: conditions under which the dialogue ends.

Dialogue games that base their definition on pre- and post-conditions of each game move are called axiomatic. There are private and public axiomatic dialogue games, and their semantics is based on internal and observable propositions respectively (McBurney & Parsons 2003).

Dialogue games have been used mainly in linguistics, therefore their rules are based on beliefs and intentions of the agent that performs the speech act in order to explain how decisions are made, characteristically they have been private axiomatic dialogue games. This differs from our objective, since our interest is on modeling protocols from a global perspective and not from the perspective of any of the participants. Our objective is to define game rules and verify conformance to them based only on the visible part of the dialogue. In a scenario that involves autonomous agents, it is not an option to verify if an action follows the internal rational state of an agent.

3.5.9 OWL-P

Based on theories developed by Singh and his group about Commitments (Mallya & Huhns 2003, Desai & Singh 2007), a model for protocols is proposed from the engineering point of view (Desai et al. 2005*b*). The objective is to support:

- Openness of the system in order to make it flexible.
- Autonomy of the participants, so that the course of actions can be produced by them
- Exceptional situations, even though the main focus is on the most important and common actions.

These objectives are approached by introducing refinement in protocols: protocols are specified in such a way that they model the main desired course of actions, later some parts of it can be replaced by other protocols that serve as refinement, adding more details of how the process is concretely done.

Semantics are given to protocols in terms of Commitments. These are coded by using an ontology for protocols written in OWL (Web Ontology Language). Composition of protocols is done by making use of *compositions axioms*. Composition axioms state how protocols are to be connected together, they can be:

- Role Definition Axioms: connects roles between protocols to a single role in the composed protocol, in the sense that they will be played by the same agent.
- Data Flow Axioms: connects data slots together, by saying which slots in the first and the second protocols are the same and will be seen as one in the resulting composed protocol.
- Implication Axioms: tells what assertions in one protocol imply in the second protocol.
- Event Order Axiom: specifies ordering among messages of the protocols.

They introduce the concepts of closed and open protocols. If all slots are defined and all Commitments are fulfilled the protocol is closed, otherwise the protocol is open.

Role skeletons are produced out of the protocols modeled with OWL-P. These are the definition of the view of each role, a summary of the expected visible behaviour of each role. Agents playing those roles will augment them with their policies to provide the necessary business logic.

Protocols are described based on propositions used to define states. The descriptive nature of this approach provides some advantages, for instance, these propositions can be used to define constraints that represent preferred paths in a protocol (Mallya & Singh 2006*a*), by telling what "events" must have happened (or what propositions must hold) at the end of the protocol enactment. These constraints represent groups of runs that fulfill that constraint. These groups of runs can be organized in hierarchies to represent the order of preference among possible runs.

Even though OWL-P has a declarative approach, it still has hard sequencing of messages (Desai & Singh 2007), restricting the places where a protocol can be extended. Sequencing is done by connecting messages directly and not by means of an abstraction that provides mechanisms for protocol modification or composition.

3.5.10 AMOEBA

Further development of OWL-P and Commitment algebra lead to AMOEBA (Desai, Chopra & Singh 2009), a Methodology for Modeling and Evolution of Cross-Organizational Business Processes. It treats interaction at the level of business semantics and not just at messaging level. UML diagrams extended to use Commitments are used to represent protocols based on the semantics in OWL-P. Using Commitments expresses the business semantics of the protocols better in comparison to common approaches. AMOEBA makes enactment and requirement adaptation of protocols flexible.

AMOEBA highlights the importance of the work done at business semantic level and focused on the publicly visible part of the interaction: the internal business logic of each participant is not reusable, since it reflects the internal interests of the participant and will hardly match those of another one. Interaction patterns, on the contrary, have as one of their main reasons to exist, to be reusable.

The amoeba methodology consists of the following steps:

1. Identify Roles and Protocols: identify all participants in a process and abstract roles out of them. Also, group all logically related interactions and associate each group to a protocol.

2. Identify Contractual relationships: represent these relations in terms of Commitments. They can be those existing before the interaction is started or those that started during interaction enactment. Specify how these Commitments are created (for the second ones), manipulated and discharged.

3. Specify message meanings: messages are to be defined in terms of their effects on conditions and Commitments using non-monotonic causal logic.

4. Specify constraints among messages: add message ordering based on data flow or temporal constraints.

5. Compose protocols to make the business process: compose the protocols together to represent the business process again. This requires roles in each protocol to be associated to those in other protocols that are to be performed by the same participant. Connect data flows and constraint events in order to define the sequencing of the protocols. For this purpose, role, data flow and event axioms are created.

Special interest has been dedicated to support adaptation and evolution using this methodology. Processes in amoeba can be adapted by taking advantage of protocol composition. If a change is required, only those protocols affected are modified using a procedure similar to the steps described here for creating business processes, keeping the effects of changes to the necessary minimum.

Amoeba is based on the work in OWL-P and therefore shares the same properties related to our objectives. It shows the advantages that a declarative approach provides in terms of flexibility to modeling business processes. Still there are many details that are left for the designer to define instead of representing them in the model, like message sequencing. This constrains the possibilities of AMOEBA to provide helping features at the time of composing protocols. Our objectives are very similar, but aim also at representing explicitly how protocols can be composed and support the composition phase.

3.5.11 Goal-oriented definition of protocols

Goals are concepts that represent the objective of some effort. In the area of multi-agent systems, goals are used to declare the purpose an agent is pursuing or to guide the implementation or execution of plans and actions in agents. Braubach and Pokahr (Braubach & Pokahr 2007) have defined goal-oriented interaction protocols. They have augmented AUML interaction representation, used as domain-independent layer, with domain layers associated to each role in the diagram. The domain-independent layer is the globally visible actions in the form of a protocol and the domain-dependent layer is the set of behaviours carried out inside agents. They specify goals for the domain layer that arise and initiate agent actions that make them behave corresponding to the protocol.

In agent implementation tools, goals are very important. For instance, they are fundamental for BDI agents like JACK (Winikoff 2005) or JADEX (Pokahr, Braubach & Lamersdorf 2005*a*). Goals are used to define or even implement agents, therefore they are normally specified from the perspective of the agent. This is the case in dialogue games and FIPA speech act library: internal goals are what motivates agents to perform actions or they are at least used to define preconditions of actions.

The use of goals as usual in multi-agent systems is not helpful for the specification of interaction protocols, which are required to be defined from a global perspective. Even so, the concept of goals would bring some benefit, if they are defined in the same terms of the protocol. Goals can be used to characterize protocols in order to classify them and provide support in tasks like choosing and comparing protocols, in a similar manner as agents use them to reason about actions or plans to take. This is an advantage of declarative definition of goals (van Riemsdijk, Dastani, Meyer & de Boer 2006).

3.5.12 Service oriented architecture Modeling Language (SoaML)

Service Oriented Architecture (SOA) is a way of organizing and understanding organizations, communities and systems to maximize agility, scale and interoperability. Services in this architecture are capabilities offered through well-defined interfaces to the community. SoaML is intended to architect and model SOA solutions by extending UML. The approach in SOA separates the concern of *what* needs to be done from the *how, where* and *who*. It can be used for basic stateless services, but also to enable organizations to cooperate in a community using inter-related sets of services. It can be used to specify the participants of a community and the services contracts that tell how interaction is to be done to achieve some specific purpose.

SoaML takes advantage of Model Driven Architectures (MDA) to ab-

stract the logical design of a SOA architecture from any possible concrete realization, a key requirement to support the wide variety of technology used to realize these architectures. It introduces a series of concepts:

SoaML Concepts

In order to model SOAs, a set of concepts are defined. These, together with the relationships between them, compose the meta-model for SOAs proposed by this modelling language. These concepts are:

Service: A concrete *capability* offered by some entity or entities which is described precisely enough, for it to be used by other requesting entities. These entities, providers and requesters of services are called *Participants*.

Participant: entities taking part in a community described by a SOA is called a *Participant*. Participants provide services using a *service port*.

Service Port: service ports are the interaction points provided by service participants to let others use the service. Service ports are typed using *Service Interfaces*.

Service Interface: type definitions used to describe how to use a service port. It is the contract that specifies how to interact with the service. It is defined from the perspective of the service provider and it contains:

- The provided and required Interfaces: specify the messages that will be received and sent and the provided capabilities.
- The enclosed parts: the roles involved in the service. Roles typed by the realized interface is played by the provider and the role typed by the used interface is played by the consumer.
- The Behaviour: rules which interactions are valid, plays the role of a communication protocol.

Capability: is an Interface realized by a *Service Interface*. It is an abstraction of the services of some system regardless of the participants that are using it.

Service Capability: a cohesive set or functions or capabilities that a service might offer.

Protocol: is the order in which internal interfaces of a *Service Interface* are organized. It is an owned behaviour of the *Service Interface*.

Role: definition of the basic functions a participant must have in order to perform in a specific context.

Collaboration: the modelling of a services architecture of a specific community of participants. This architecture is defined in terms of Roles which will be played by the participants.

Service Contract: definition of terms, conditions and choreography that participants, providers and consumers, must agree to. Participating Roles and their requirements are defined by Service Interfaces.

Choreography: is a specification of what and when is transmitted between participants, the obligations that go between parties to enact a service exchange. The definition of a choreography is always limited to what happens between participants avoiding their internal processes.

SOA-ML is a recent contribution that shows how a model of a business process can be used in practice. It also shows the adequacy of using MDA in a scenario where some concept is developed in an abstract level and is to be transferred automatically to concrete an implementation. UML sequence diagrams are used for representing the actions in an interaction, having SoaML, for our work, a similar value as other contributions that use this kind of diagrams.

3.5.13 PIM4Agents

Using MDA approaches for designing and implementing business processes has shown the be important (Hahn, Zinnikus, Warwas & Fischer 2009). This also applies to multi-agent systems (MAS), specially those used to implement business processes. For that purpose a platform-independent domain-specific modeling language called DSML4MAS (Hahn, Madrigal Mora & Fischer 2009) has been developed. Like any language, DSML4MAS consists of an abstract syntax, formal semantics (Hahn & Fischer 2008) and concrete syntax (Warwas & Hahn 2009).

The abstract syntax of DSML4MAS is defined by a platform independent meta-model for MAS, called PIM4AGENTS, defining the concepts and their relationships. The core of the PIM4AGENTS is structured into different viewpoints briefly discussed in the remainder of this section.

- *Multi-agent view* contains the core building blocks for describing MAS. In particular, the agents situated in the MAS, the roles they play within collaborations, the kinds of behaviors for acting in a reactive and proactive manner, and the sorts of interactions needed for coordinating with other agents.

- *Agent view* defines how to model single autonomous entities, the capabilities they have to solve tasks and the roles they play within the MAS. Moreover, the agent view defines which resources are accessible for the agent and which kind of behaviors it can use to solve tasks.

- *Organization view* defines how single autonomous agents are arranged to more complex organizations. Organizations in DSML4MAS can be either an autonomous acting entity like an agent, or simple groups that are formed to take advantage of the synergies of its members, resulting in an entity that enables products and processes that are not possible for any single individual.

- *Role view* covers the abstract representations of functional positions of autonomous entities within an organization. In general, a role in DSML4MAS can be considered as set of features defined over a collection of entities participating in a particular context. The features of a role can include (but are not be limited to) activities, permissions, responsibilities, and protocols. A role is a part that is played by an entity and can as such be specified in interactive contexts like collaborations.

- *Interaction view* focuses on the exchange of messages between autonomous entities. Thereby, two opportunities are offered: (i) the exchange of messages is described from the internal perspective of each entity, or (ii) from a global perspective in terms of agent interaction protocols focusing on the global exchange of messages between entities.

- *Behavior view* describes the vocabulary available to describe the internal behavior of intelligent entities. The vocabulary can be defined in terms of combining simple actions to more complex control structures or plans that are used for achieving predefined objectives or goals.

- *Environment view* contains any kind of Resource (i.e. Object, Ontology, Service etc.) that is situated in the environment and can be accessed and used by Agents, Roles or Organizations to meet their objectives.

- *Deployment view* describes the run-time agent instances involved in the system and how these are assigned to the organization's roles.

To lay the foundation for further discussions on how to use DSML4MAS for modeling interactions, we focus on the interaction viewpoint in the remainder of this section. Key PIM4Agents concepts for the present work are illustrated in Fig. 3.12.

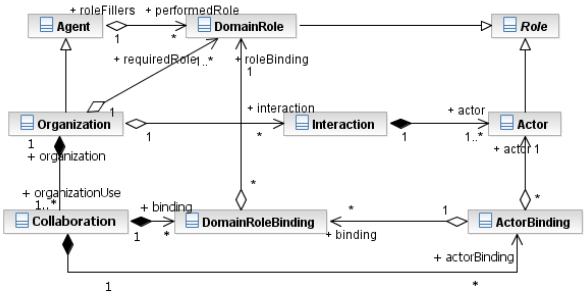


Figure 3.12: Key Concepts in PIM4Agents.

The interaction aspect of DSML4MAS defines in which manner agents, organizations or roles interact. A *Protocol* is considered as a special form of an *Interaction*. In the deployment view, furthermore, the system designer can specify how *Protocols* are used within *Organizations*. This is done through the concept of a *Collaboration* that defines which organizational members (which are of the type *AgentInstance*) are bound to which kind of *Actor* as part of an *ActorBinding*. Beside a static binding, the designer may want to bind the *AgentInstances* at run-time.

An interaction protocol as a pattern for conversation within a group of agents can be more easily described using generic placeholders like 'Initiator' or 'Participant' instead of describing the interaction between the particular agent instances taking part in the conversation. In DSML4MAS, this kind of interaction roles are called *Actors* that bind *AgentInstances* at design time or even at run-time. Furthermore, *Actors* require and provide certain *Capabilities* and *Resources* defined in the role view of PIM4Agents.

Messages are an essential mean for the communication between agents in MAS. In PIM4Agents, there are two sorts of messages, i.e. *Message* and *ACLMessage* which further includes the idea of *Performatives*. Messages have a content and may refer to an *Ontology* that can be used by the participating *Actors* to interpret the *Message* and its content.

PIM4Agents has grown to be a very comprehensive MDA tool for developing multi-agent systems. Several mappings have been implemented to different concrete multi-agent systems tools like Jack, Jadex or Jade (Warwas, Hahn & Fischer 2009).

This MDA approach has proved to contribute significantly to the multi-agent community in the area of business process modeling and implementation. Its modeling of interaction protocols inherits many concepts from AUML and similar models (Sections 3.5.6 and 3.5.7). It has many enhancements to couple interaction protocol models to participant behaviour models and also to enclose these protocols in the context of an organization. Our

contribution seeks to enhance these protocol models in order to provide tools that aid their development and at the same time to provide a platform that allows to run processes developed with this tool using Web Services.

3.6 Summary

Work related to our topic has been reviewed at two levels: first, Web Services and FIPA-agents integration and second, complex interaction models for implementing business processes in the areas of Web Services and agents.

The review compares and shows that Web Services communication and FIPA specifications are complementary and very suitable for integration. FIPA communication is focused on the complex levels, like interactions and Web Services provide a very concrete structural basis. FIPA specifications stack provides freedom of grounding. Web Services is the preferred system integration technology and the FIPA stack can be grounded on them.

The multi-agent community has provided several tools to integrate FIPA agents and Web Services, most of them are based on Gateway approaches that do not support complex conversations and don't take advantage of the messaging architecture of the agent platform. The FIPA Agents and Web Services Integration working group has proposed our communication framework as a suitable way to integrate Web Services and FIPA.

A comprehensive set of models and tools for creating and implementing business processes and interaction protocols has been reviewed also, from very basic but widely deployed frameworks like RosettaNet to very new MDA multi-agent system tools like PIM4Agents. Web Services approaches have failed in that they miss fundamental features necessary for modeling, like multi-casting and management of cardinality of participants. Some approaches are just not oriented to the solution we are looking for, like WS-BPEL, which is intended to orchestrate Web Services and execute these orchestration descriptions as if it was a computer program. Others like, WS-CDA try to cover a modeling problem with techniques suitable for implementation. UML has provided tools for modeling interactions which have shown to be somewhat suitable. Some of their disadvantages have been covered by modeling tools for agents like AgentUML. Still, this way of modeling has some inconveniencies related to modularity and recombination. OWL-P and the AMOEBA methodology shows that a declarative approach provides improvements in flexibility and maintainability, fundamental for business processes, still their declarative approach does not cover all concepts of the model, making their models half declarative half imperative. In the area of multi-agent systems, other approaches for declarative definition of dialogues have been developed, like goal-oriented protocols or dialogue games. These approaches also prove the advantage of a declarative model, but base their definitions on the perspective of the participants. It is

fundamental for our work to use a global perspective that defines interaction protocols independently of the perspective of the participants.

Our group has developed in the last years MDA tools for multi-agent systems called PIM4Agents. These tools provide a comprehensive set of features that can be used to model business processes and already are capable of generating executable code based on the models. PIM4Agents will serve as a hosting framework for our proposed meta-model for interaction protocols.

Part II

Solution

We have seen that the problem to integrate systems via communication is twofold: first, a low level integration of FIPA agents and Web Services is pursued and then, an improved model for interaction protocols is to be proposed. In the next two chapters, our solution for these two main problem areas will be presented.

First, in the next chapter, a message representation that can work well with the semantics of both specifications: W3C Web Services and FIPA agents will be produced. The proposed specification is compliant in both scenarios.

In the chapter afterwards, a protocol specification model will be presented in a formal manner, in order to explain in detail how different contributions of multi-agent systems are going to be put together. The main goal of this specification is to have a protocol model that supports modularity, similarly as proposed by (Desai, Mallya, Chopra & Singh 2005*a*), in which a deeper semantic definition of actions allows to produce explicit definitions of protocols.

Chapter 4

Integrated Messaging Architecture

Taking into account Web Services and FIPA specifications, their properties, features and goals as discussed in chapter 3, an integration will be proposed which consists in a different grounding for FIPA specifications. A messaging stack is proposed for allowing FIPA messaging over Web Services that connects both specifications at the message envelope level (León Soto 2006). Doing this, FIPA agent platforms would be capable of communicating over Web Services standards and integration will be possible.

The main objective is to produce a *Web Services based Message Transport Service (MTS)* that enables agents to interact through the Web with other Web Services and agents. FIPA communication framework must remain the same, with the only difference that the grounding of the messages must use Web Services standards. Agents that can communicate using the infrastructure provided by Jade without using the add-on, will be able to communicate using the add-on on Jade, it will not require significant changes in their implementation.

Other objectives are:

- **Accessibility:** agents must be capable of connecting to Web Services using the same mechanisms for communicating with other agents. Agents must also be accessible by conventional Web Services. Accessibility must be possible not only with WS-Addressing compliant Web Services, but also with simple or REST services.

- **Web Services compliance:** Jade should, with this add-on, use conventional Web Services standards, particularly SOAP and WS-Addressing. This is important to ensure that any contribution achieved with this tool works appropriately in any Web Services scenario.

- **Enable complex interaction patterns for Web Services:** Web Services are used dominantly in interaction patterns similar to RPC.

An increasing interest exists on supporting more complex interaction patterns, specially in areas like Business Process or work-flows enactment. This tool should enable the implementation of such conversations: using Jade to implement FIPA interaction patterns but grounding them using Web Services standards.

- **Integration of Jade into a Web Services infrastructure:** Jade should not be used for development only, but also for performing as any other system in a Web Services environment. Therefore it must be integrated as any other conventional java Web application.

- **XML Content Description:** Jade provides mechanisms for creating ontologies, that can be used in the contents of agents' knowledge and also for the contents of messages. Jade will be extended with a XML grammar for the SL language in form of an XML schema.

4.1 Foundation for integration

The architecture chosen as a solution in this chapter is based on some properties of the areas to be integrated. Section 3.3 gives a comparison of Agents and Web Services in terms of some aspects relevant for the present work. In that section, the comparison is used to show why Web Services and Agents integration is important (See Table 3.7). In this section the same comparison will be as a foundation of the proposed architecture.

First of all, it is important to remark that FIPA specifications stack has done an important amount of work specifying semantics in form of abstract specifications. This is explicitly done so in order to leave room for different ways for grounding. FIPA messaging stack (Figure 3.5) shows how a FIPA ACL message (abstract) specification is placed on top of an envelope(abstract) representation which can be implemented (concretely) as desired. Some groundings for these specifications are provided by FIPA, but these are limited to follow the outline of FIPA's abstract specification and use a LISP-similar syntax that has little use in other areas. Web Services on the other hand are more about the grounding. Specifications are mainly focused on how to represent things "on the wire". This situation is the main advantage used by the proposed architecture to integrate both areas. Making use of the crucial separation of concept and representation in FIPA and its intention to support and have multiple ways of representing the specification's concepts, a new way of representing them will be defined, this time using widely adopted Web Services Standards. At the same time Web Services do not impose semantic constraints and structures in their specifications. Even though both areas can be seen as being very different, as a matter of fact, as shown here, they complement each other in a very suitable way.

Even so, there are some details that need to be cleared in order to define how FIPA messages will be represented using Web Services. In this case one can see that Web Services specifications, specifically SOAP and WS-Addressing, are almost a subset of FIPA's message envelope specification. The main difference in the fields that both have in common is cardinality. For instance, receivers in FIPA can be an arbitrary amount (to support broadcasts), but in Web Services there is always one specific receiver of the message. In cases where more than one receiver is set, what tools that support this do in the background (for instance Jade) is to produce one message for each of the receivers in the list. Fields that exceed what is specified in WS-Addressing make use of the extensibility provided by this standard to add extra (application specific) fields. In this case, this information is specific to agent applications. This way, the rest of receivers that was mentioned in the FIPA message are added in an extended field in the message. A Web Services infrastructure does not use them anyway, but doing so makes sure that a FIPA compliant receivers can have this information available, as it would have, if it had not used Web Services based communication.

As it has already been mentioned, this integration is based on communication only. Some of the other approaches done so far turn up to be more complex because they include description and discovery of services in the mapping. In this approach, descriptions will be regarded as one more kind of content of messages and service-directories, like UDDI, as one more kind of Web Service. Doing so separates these aspects from the problem of communication which is the core problem in interoperability. Hence, a mapping of descriptions (WSDL and FIPA service descriptions) is not part of the present work for the following reasons:

- Service publication and discovery, even though they are so fundamental in a SOA are after all just a specific application of Web Services invocation. Registries are a specific kind of Web Service and publication and discovery are simply the action of communicating with them. Therefore solving communication will provide improvement in this area.

- Description mapping between the two areas does not enjoy the synergy found in communication level. Therefore a mapping would have to make many compromises. An approach in which agents use Web Services based discovery mechanisms directly is preferred. These could be simple WSDL-UDDI up to more complex Semantic Web Services approaches.

- Publication and discovery are not part of the present approach for realizing business processes. Section 1.3 explains why.

- There have been several other contributions in this area. Chapter 3

includes a review of many projects that have done this kind of mappings.

4.2 FIPA Message Envelope using WS-Addressing

The most important detail about the FIPA-WS Stack is the representation of a FIPA Message Envelope using WS-Addressing and SOAP. Figure 4.1 shows a mapping from Web Services to FIPA messaging and Figure 4.2 shows the inverse mapping. They have two columns listing field names on each side. On the right side are the FIPA Envelope Fields and on the left side the fields of a WS-Addressing envelope. WS-Addressing allows to extend the properties set with *reference parameters*. The left side makes use of that and adds some fields to hold information present in the FIPA Message Envelope Specification. Fields belonging to WS-Addressing are marked with a continuous frame around them and have the prefix *wsa*. Extensions to hold data belonging to the ACL specification are underlined and those belonging to FIPA Envelope specification are framed with a dotted line.

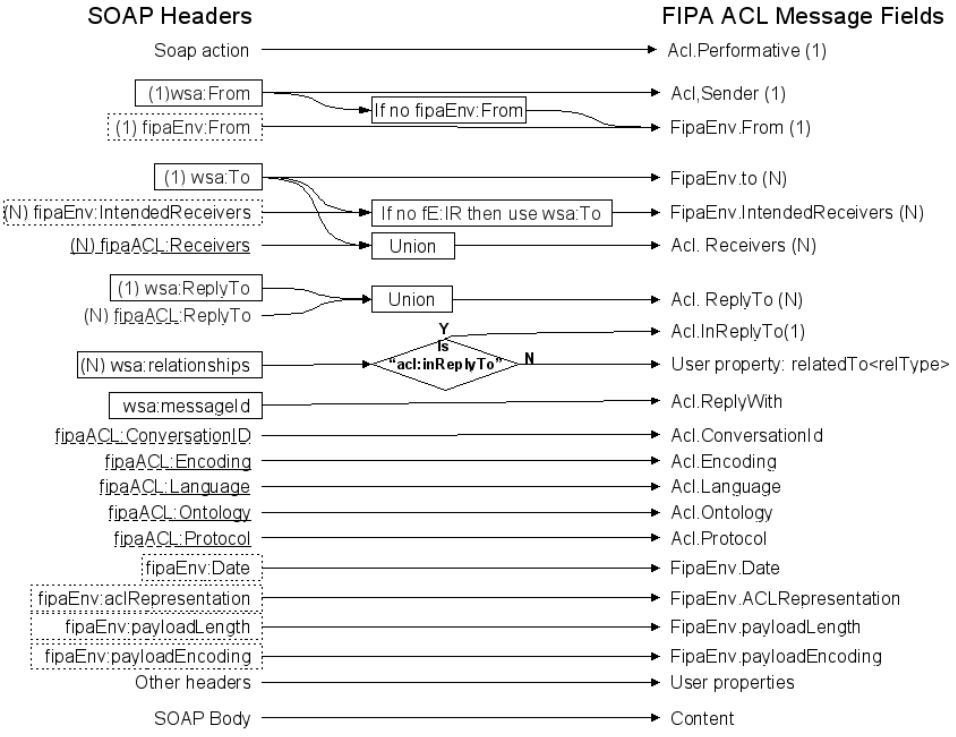


Figure 4.1: A WS-Addressing to FIPA Mapping

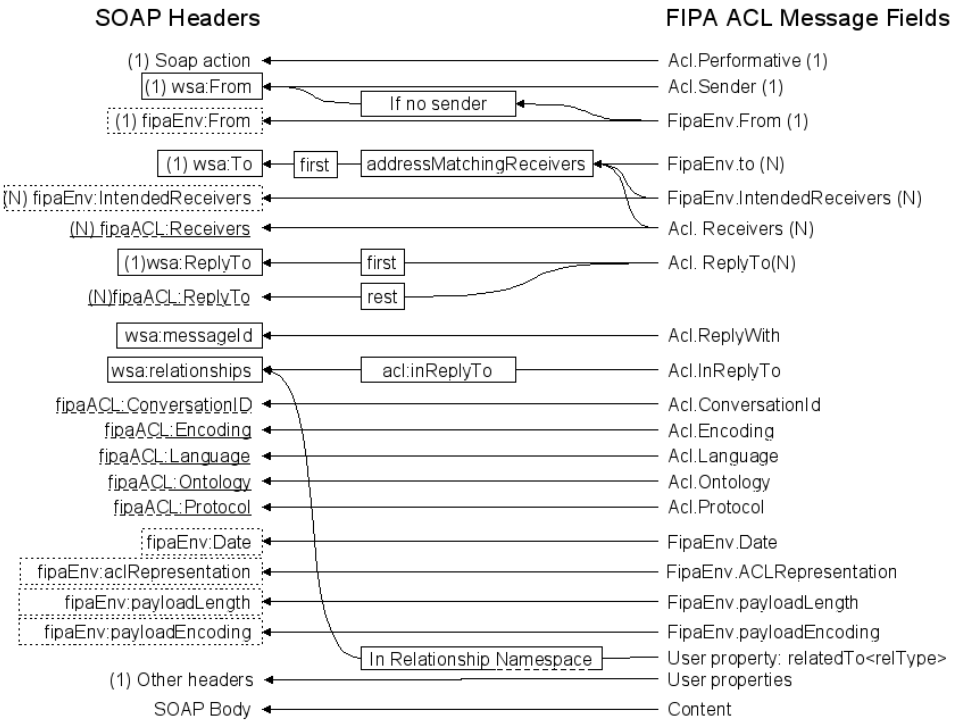


Figure 4.2: A FIPA to WS-Addressing Mapping

In the middle there are associating arrows that represent a mapping between the fields in each specification. Most of them are straightforward, but some reflect the adaptations that were necessary to produce a bijective mapping between the two specifications. `wsa:from` populates `FipaEnv:From` in case no such a field is present in the Web Services message, this is mapped backwards by populating `wsa:from` with `FipaEnv:From` in case no value in `Acl:Sender` is present.

FIPA messaging has, in contratst to WS-Addressing, a very complicated management of targeted participants for a message. Three fields are used to manage this information, each with cardinality `N`. Therefore the corresponding message properties were added in order to keep this information in the message. The value in `wsa:To` is used in `FipaEnv.IntendedReceivers` in case no such a field was present in the Web Services message. `Acl.Receivers` simply collects a union of its corresponding field in the Web Services message plus `wsa:To`. The reverse is done a bit different, each field populates its corresponding field in the Web Services message and from all of them, one is selected following some criteria to populate `wsa:To`. This criteria is defined based on the way the mapping is invoked for the ACL message. This invocation requires a target address, and the first recipient matching

this address is used as receiver of the message. This mapping of receivers already takes into account some constraints set by FIPA to establish the relation between the 3 fields: `FipaEnv:to`, `FipaEnv:IntendedReceivers` and `ACL:Receivers`. It is in conjunctions with these constraints that this mapping behaves bijective regarding receivers as well.

WS-Addressing, as stated in its name, is targeted to support messaging based on the address of the Endpoint. It does not take into account for any purpose, the identity of the agent behind the endpoint. It is therefore not possible to ensure using WS-Addressing only, that the same agent instance will be targeted at all times using the same Endpoint-Reference. On the other hand, FIPA messaging mandates to specify the ID of the targeted agent in the envelope. Therefore the context of the multi-agents application and the conditions in which the communications will be performed will be relevant for the definition of such a mapping for an agent platform. This issue will be discussed further in section 9.1.

The fields to specify where to reply to have a modification to adapt the difference in cardinality: `wsa:ReplyTo` has cardinality 1 and `ACL.ReplyTo` has cardinality N. This is adapted by adding an extra messaging property to hold the exceeding values in `ACL.ReplyTo`.

FIPA takes only into account a direct relationship between a message and its reply using the field `ACL.InReplyTo`. Web Services addressing foresees for this and any other relationship between a message (it does not have to be always a *reply* relationship) the field called `wsa:relationships`. In case a relationship expresses that the message is a reply to another message, this is set in `ACL.InReplyTo`. Otherwise any other relationship is added as a `user property` which is the only extension that was necessary in the specification of FIPA envelopes. FIPA uses another field called `ACL.ConversationID` for a similar purpose. It, as the rest of the fields, is mapped as an extension to WS-Addressing.

In order to populate endpoint fields like `wsa:To` an *Endpoint Reference* (EPR) is necessary. This is mapped to a Fipa AgentID as shown in Figure 4.3. Again, the extensibility of WS-Addressing is used to produce an AID representation based on the specification for an EPR. Reference parameters are added to include additional addresses, the agent name and any resolver defined in the Agent ID.

This mapping tries to match as precise as possible both specifications. It makes clear where the differences are and solves them by making use of the extensibility of WS-Addressing. WS-Addressing is intended to be used this way, as a common base for messaging in the Web, leaving free room for applications to extend it as they require. This mapping does so by making proper use of the few requirements that WS-Addressing does, letting agents make use Web Services compliant messaging infrastructure, since the messaging information provided by them is appropriately represented using WS-Addressing specifications. On the other side, any party interested in

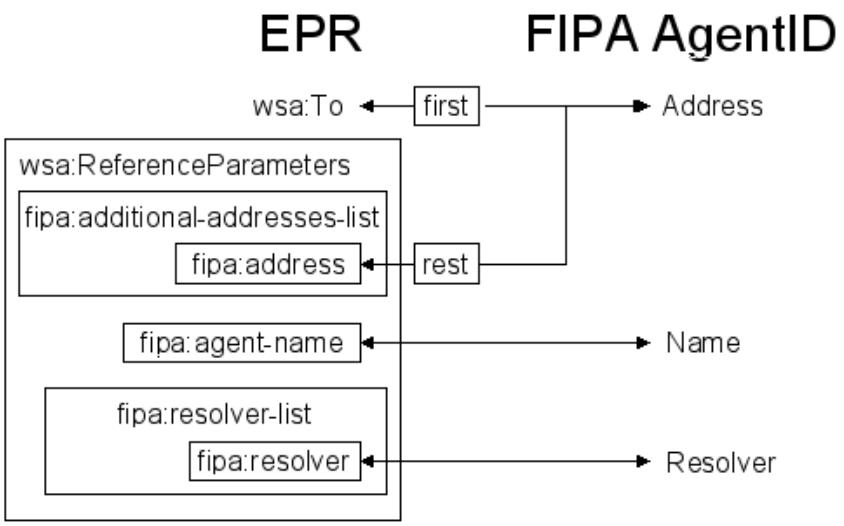


Figure 4.3: An Endpoint Reference to Agent ID Mapping

contacting an agent can do so by following the description this agent can publish. This description will include the additional fields an agent platform will demand, so that the party can add them as expected. This is the same way any other Web Services application would proceed concerning its extensions.

4.3 FIPA-WS Messaging Stack

The proposed messaging stack is shown in Figure 4.4. At the very bottom is the basic transportation layer. This layer is composed of the different network transportation protocols already used in both architectures. On top of that, the XML-based Web Services messaging is implemented using SOAP (W3C 2003) which is a service oriented messaging specification very similar to messaging in multi-agents systems. WS-Addressing (W3C 2006*c*) is used as the Web Service standard envelope specification. Based on it and on the FIPA Message Envelope specification an envelope structure is proposed as an union of both specifications. The remainder of FIPA communication stack rests on top of this new messaging specification: content specification and Interaction Protocols.

4.4 Architectural integration

Finally, having a successful stack implementation and a suitable WS-Addressing mapping, it is possible to provide an agent platform that communicates

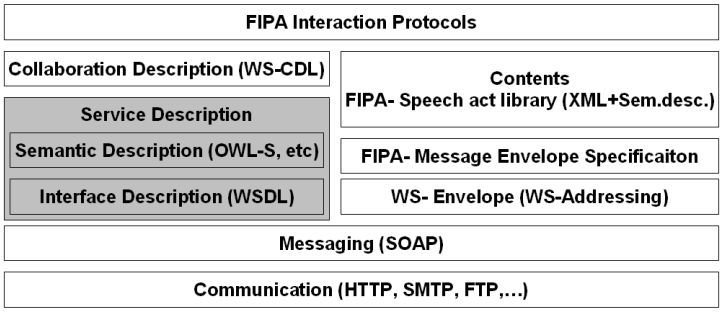


Figure 4.4: FIPA Communication Specifications Stack

using Web Services. It allows to address agents on remote platforms and perform complex dialogs with them in the same way as done currently using the existing FIPA groundings.

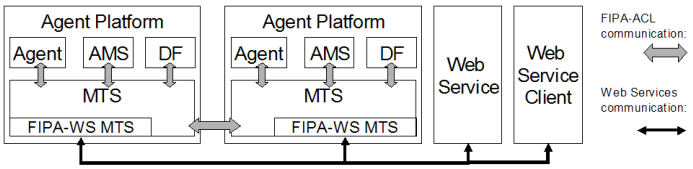


Figure 4.5: FIPA and Web Services architectural integration

This WS-grounding allows seamless communication with other Web Services (clients or providers). Agents can offer a specific service providing a description of the messages to be expected and the possible responses in form of WSDL descriptions. Figure 4.5 shows this conceptual architecture. It shows two agent platforms that have been enhanced with a FIPA-WS MTS. This extension provides the connection (shown as thin black arrows) to other remote platforms and conventional Web Services clients and providers. The original communication provided by the agent platform remains available (wide gray arrows). In this figure, it is clear that to communicate with remote agents, the traditional way as well as the way based on Web Services are available and both achieve the same results.

4.5 Summary

This section describes how both messaging architectures are integrated. FIPA provides the possibility of extending its specification with new groundings. Our integration is implemented therefore as a new grounding of FIPA specifications using Web Services: FIPA ACLMessages are represented using WS-Addressing Envelopes. Message properties of both specifications are

kept in our solution. A detailed bijective mapping between WS-Addressing Envelopes and FIPA ACL-Message and Envelopes is provided. Gaps between specifications are fixed using the extensibility facilities provided in both specifications.

A difference with other Agents-Web Services integrations is that it approaches the communication integration only, mapping of descriptions and discovery mechanisms, approached by other solutions, are left out in ours. Instead, our solution is the only one that allows to perform complex conversation patterns.

Our messaging stack lets FIPA agent platforms to communicate over Web Services enabling transparent interaction between Agents and Web Services. FIPA communication mechanisms are kept unchanged and, at the same time, communicating with WS-Addressing and conventional REST services is made possible. Our solution results in an architecture that seamlessly enables communication between parties of both specifications.

Chapter 5

Protocol specification

The model of a business process specifies the order in which the activities are to be performed. (Weerawarana et al. 2005) calls this prescription the *control flow*. This control flow has direct influence on very important properties of a business process like its cost and duration. In this section a model for specifying control flows will be presented in the form of an interaction protocol.

A clear contract between participants of a collaboration is a crucial element in order to achieve true integration. As shown in section 3.2, the multi-agent community has developed different approaches for creating an interaction protocol model, but has not been successful in the sense that there still does not exist a comprehensive, formal and concrete model that supports development of interaction protocols in a modular and reusable manner. Section 3.5 shows that as a matter of fact, even AgentUML, an enhanced model for FIPA interaction protocols and the preferred modelling language used for interaction protocols, lacks a semantic connection between the Speech Acts and the protocol that contains them. There is no explicit explanation of why a protocol is composed of a certain sequence of actions which at the same time leaves no way for justification of why a protocol can be composed with another one or why a sub-protocol can be inserted into a specific section of another protocol. Both situations are handled solely by the rationality of the observer and his understanding of the meaning of actions.

This chapter proposes a model that serves three purposes:

- To provide an explicit modularity model by connecting the definition of a Speech Act and that of a protocol.
- To consolidate the various aspects that have been contributed by the multi-agent community in a single model.
- To disambiguate concepts that have been used differently among contributions reviewed in Chapter 3 and to establish a concrete and de-

tailed definition of these concepts and how they relate to each other.

Therefore, a model of interaction protocols in the form of a *contract* between several participants will be specified as the description of interactions from a *global* perspective. It is a description of the possible actions the different participants can perform given a state of the conversation and how these actions affect the global state of the conversation.

The model is based on planning models for agents, but from a different perspective. Conventionally, planning systems use action descriptions, based on propositions, to produce plans and achieve a goal. The central concept in the model being proposed here is to have prefabricated plans (interaction protocols) and use the action description to explain the structure of protocols in the same way that actions define plans in conventional planning. Using the same planning metaphor, interaction protocols can be composed having explicit semantics of why and how one protocol can follow or replace another (sub)protocol.

In the present approach, a protocol is seen as a transition system (Mallya & Singh 2004) very similar to a finite state machine (FSM): conversations are modeled as a set of states connected by actions. States are defined by a set of propositions with assigned truth values. Actions change the truth value of some specific propositions to reflect the effects in the conversation moving it from one state to another. In the context of business processes, these conversations can involve a considerable amount of actions and states, if the protocol is designed to be flexible, complexity increases to allow more alternative paths and states. In consequence, a FSM that represents complex flexible interaction processes are susceptible to becoming large and hard to manage. It is mandatory to reduce complexity in every possible way. One of the preferred ways to cope with complexity in software engineering, and paradoxically also in multi-agent technology also, is modularization. Therefore, the main objective of the present model is modularity.

In the following sections the different components of the protocol model will be formally defined. First the state-space and state-descriptions are defined, the second one being a simpler way to refer to many specific states that have some properties in common. Afterwards actions are defined as operations that change conversations from one sate to another. Having this basic state-action model, the different aspects required specifically by interaction protocols and that differentiate them from simple agent plans, are defined: roles, time management constraints (Timeouts), commitments and other special kinds of propositions. The last component necessary for this model is cardinality constraints on different aspects of an interaction protocol.

These fundamental concepts are used firstly to define what an interaction protocol is and secondly to allow the discussion of some properties and usual patterns. Finally composition of protocols that follow this model are

explained, showing why this model supports modularity. A brief example shows how these concepts work.

5.1 Brief definition of a state-action space

The foundation of a model based on actions as transitions between states is the state-action space. As with FSM and classical planning, we need a way of describing situations and how to go from one to another. Similar to classical planning, we will work with states defined by truth values associated to propositions. For the model that will be defined, the state-action space is composed of State Descriptions and Speech Acts. A thorough definition and discussion about this model can be found in (León Soto 2009). A summarized definition is as follows:

5.1.1 State Descriptions

As a method of abstracting to what is only relevant for the definition of a protocol, the concept of a State Description is defined. A State Description, as its name says, describes what propositions are necessary to be known about a concrete state.

Having a set P of all the *proposition names* used in the system to describe states and using the syntax "p" and "$\neg p$" to say $p \in P$ is a valid or invalid proposition respectively, State Descriptions are defined as a set of concrete states that match certain constraints on some of their propositions:

Definition 1. *A State Description* s *is the set of all states that match all the constraints of the state description. These constraints are represented as some propositions* $[\neg]p_i, p_i \in P$ [1].

$$s([\neg]p_a, \ldots, [\neg]p_b) = \{\sigma \in \Sigma \mid [\neg]p_a \in \sigma, \ldots, [\neg]p_a \in \sigma\} \tag{5.1}$$

where:

Σ*: set of all concrete states in the system*

$[\neg]p_a, \ldots, [\neg]p_b$*: propositions on some arbitrary proposition names subset of* P*.*

The set of all state descriptions is called S*.*

5.2 A model of Speech Acts

The other component necessary to complete our state-action model are Speech Acts which represent the action of an agent sending a message to another one. A conversation involves always at least two participants; these participants are represented as *roles* (r) which are members of a set called

[1] $[\neg]p$ means either p or $\neg p$.

R. Every action is always performed by a role and is targeted at another role.

Actions change the state of the conversation, and this change is specified as a set of operations on proposition names. There can be two operations: either + or − and mean, respectively, to bring about or to negate the associated proposition. For instance $+p$ turns the truth value of the proposition with the proposition name p (regardless of its previous truth value) to *true* (and correspondingly $-p$ to false):

$$+p \Rightarrow p$$

and

$$-p \Rightarrow \neg p$$

Operations allow one to calculate what State Description the Speech Act leads to, based on an enabling State Description that fits the preconditions of the Speech Act.

Definition 2. *A* Speech Act *a is a labeled association of two roles, preconditions in the form of propositions and a set of operations. Speech Acts belong the the set A.*

$$a : \langle l, r_i, r_j, \{[\neg]p_a, \ldots, [\neg]p_b\}, \{o_c, \ldots, o_d\}\rangle \tag{5.2}$$

where
l : *is a* label.
$r_i \in R, r_j \in R$
$[\neg]p_a, \ldots, [\neg]p_b$*: propositions on some arbitrary proposition names subset of* P.
$o_c, \ldots, o_d$*: operations on some arbitrary propositions.*

For instance, the Speech Act a_1:

$$a_1 = \langle \text{“}\texttt{respond}\text{”}, \mathit{seller}, \mathit{customer}, \{\mathit{quotation\_requested}\},$$

$$\{+\mathit{quotation\_provided}\}\rangle$$

represents the action labeled as “respond” that can be sent by role *seller* when *quotation_requested* is true to the role *customer* and defined as bringing about the fact *quotation_provided*. This is an action that can be performed by *seller* every time *quotation_requested* is true and always leads to a State Description where *quotation_provided* is true.

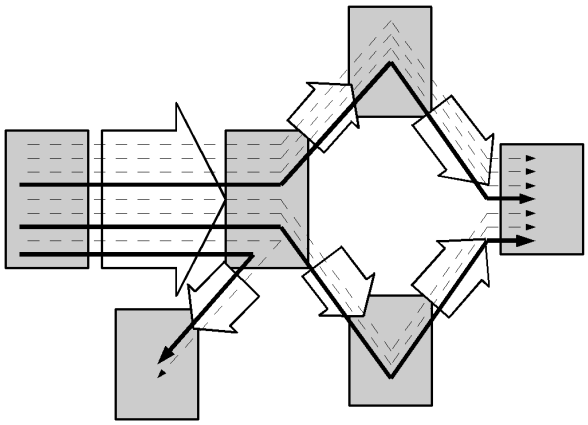


Figure 5.1: Protocol (boxes and white arrows) composed of 3 runs (thick lines) and some conversations as instances of runs (dashed thin lines)

5.3 Cardinality constraints

Protocols, in general, represent specific possible paths over the whole state-action space. Each different path a protocol can take is known as a *run*. In the concrete case of an agent performing a protocol with more than one other participant, this agent will manage one instance of the protocol for each participant. These instances are called *conversations*. Each conversation will follow its own run, some of them might enact the same run. In synthesis, a protocol is a specification of a set of runs, each of which represent, at the same time, a set of conversations. Figure 5.1 illustrates the relation between these 3 concepts.

The cardinality of runs is limited by the amount of Speech Acts that are enabled for the same State Description. The cardinality of conversations, on the other hand, is ruled by an amount that is associated with each action, called a *Cardinality Constraint*:

Definition 3. *Cardinality Constraints are two numerical values associated to each Speech Act or State Description. These values represent the minimal and maximal cardinality of conversations associated to each Speech Act* $a \in A$ *or State Description* $s \in S$ *respectively:*

$$\overset{q}{\underset{p}{}}a \tag{5.3}$$

where
$p \in \mathbb{N}$*: minimal constraint*
$q \in \mathbb{N}$*: maximal constraint*
$p < q$

$$ {}^{w}_{v}s \tag{5.4}$$

where

$v \in \mathbb{N}$*: minimal constraint*

$w \in \mathbb{N}$*: maximal constraint*

$v < w$

For brevity and agility, in cases where the cardinality of actions is free: minimum is 0 and maximum is not bound, represented with N, the cardinality constraints can be omitted:

$$ {}^{N}_{0}a = a \qquad {}^{N}_{p}a =_{p} a \qquad {}^{q}_{0}a =^{q} a \tag{5.5}$$

For instance, the following cardinality constraint means that the action called "inform" can be sent from the role r_1 to the role r_2 a maximal of 7 times and a minimum of once from State Descriptions that have $\neg p$.

$${}^{7}_{1}\langle \text{“}\texttt{inform}\text{”}, r_1, r_2, \{\neg p\}, \{+p\}\rangle$$

5.4 Special kinds of propositions

Some particular kinds of propositions are frequently needed when modeling interaction protocols. In the present model, three of them will be defined: Timeouts, Commitments and Cross-Conversational Constraints.

5.4.1 Timeouts

Protocols are mechanisms to rule actions over time. Sequencing and turn taking are problems that are solved with the present model, but there are certain cases, where concrete time-windows are to be specified. This is the purpose of defining Timeouts.

Definition 4. *A Timeout $T(t_p, a)$, where $a \in A$, is a proposition name that, when brought about (by Speech Act called a_0), states that the Speech Act a will be performed after a certain time period t_p, that starts to count after a_0 is performed.*

It is very important to remark, that:

- Speech Act a in a Timeout is not necessarily performed by the sending role mentioned in a, but instead, it can be an assumption the receiver of a can make.
- Speech Act a will always have implicitly the operation $-T(t_p, a)$ declared: Timeouts are removed automatically after a is performed.

For instance, the action a says, that after performing *call for proposals* (cfp), the role B will send to role M the message "*propose*" after a period of size t_d, after which *proposition requested (p_r)* will no longer hold:

$$a = \langle M, B, \text{“}\texttt{cfp}\text{”}, \{\neg p\_r\},$$

$$\{+p\_r, +T(t_d, \langle B, M, \text{“}\texttt{propose}\text{”}, \{p\_r\}, \{-p\_r\}\rangle)\rangle$$

5.4.2 Commitments

Singh (Singh 1999) and his group have proposed an algebra for Commitments (Yolum & Singh 2002) which allows better modularity in the design of processes (Mallya & Singh 2004), (Mallya & Singh 2005). These concepts about Commitments will be integrated into our protocol model by defining them using the syntax defined in this chapter .

Definition 5. *Commitment $C(a_d, a_c, p, c, t) \in P$ is defined as a proposition name that represents the Commitment of the* debtor *agent a_d to the* creditor *agent a_c to bring about the proposition $p \in P$ under the condition that the proposition $c \in P$ becomes true. After the condition c becomes true, agent a_d is expected by agent a_c to perform some action that produces p to be true. This action is to be performed before Timeout t, that represents the time limit, runs out of time. This Timeout starts to count as soon as the condition p is brought about.*

Furthermore, a particular kind of Commitment, an Unconditional Commitment $C(a_c, a_d, p, t)$, is an abbreviation the following notation:

$$C(a_d, a_d, p, true, t) \;=\; C(a_c, a_d, p, t) \tag{5.6}$$

It simply means that agent a_c expects a_d to bring about the proposition p within the time period specified in t and that t is already running since the Commitment was brought about.

For practical purposes, the semantics of Commitments in interaction protocols have a preponderant role. It gives meaning to intentions declared by participants. For instance, previous work about contracts between agents to solve transportation schedules (Fischer, Kuhn, Müller & Müller 1995), discusses the different aspects that can arise when a bid is placed by a candidate. Such bids are claims and they are more or less favorable for the outcome of the whole system, depending on when they have to hold: at the time of the claim?, for a longer period?, what to do when the winning bid does not hold anymore at the time of executing the offer? etc. In our work, such aspects can be expressed by the objective of the Commitments. Also, the intention to include Timeouts in the definition of Commitments is to provide the semantics of what the creditor will assume, if the Commitment is abandoned by the debtor.

It is important to note, that the Commitments are part of the set of proposition named P, they can be used to specify State Descriptions and Speech Acts. The operators + and - can also act upon them and detailed semantics of these two operations will be discussed next:

Definition 6. *Commitment creation:* $+C(a_c, a_d, p, c, t)$. *An action specifying this operation states that after the action is performed, the specified Commitment starts to exist.*

Definition 7. *Commitment cancellation:* $-C(a_c, a_d, p, c, t)$. *If the Commitment exists, performing the operation "-" on it cancels the Commitment, makes it a false proposition. After that, it does not exist anymore:* a_d *is no longer expected by* a_c *to bring about* p *and its Timeout is dissolved as well.*

Definition 8. *Commitment enablement: bringing about the condition* c enables *the Commitment. If the condition* c *is true, the Commitment is transformed to an unconditional Commitment: the conditional Commitment is canceled and the unconditional Commitment is created enabling the Timeout countdown. The following transformation rule defines the process:*

$$\frac{c \wedge C(a_d, a_d, p, c, t)}{c \wedge C(a_c, a_d, p, t) \wedge \neg C(a_d, a_d, p, c, t)} \tag{5.7}$$

Any state where a conditioned Commitment and its condition are true at the same time are automatically transformed to a state where the condition still exists, but the Commitment has been replaced by the corresponding unconditional Commitment.

Definition 9. *Commitment discharge: bringing about the commitment objective* p *before the Timeout* t *runs out of time automatically cancels the created commitments that have* p *as objective, including their Timeouts. The following transformation rule defines the process:*

$$\frac{p \wedge C(a_d, a_d, p, c, t)}{p \wedge \neg C(a_d, a_d, p, c, t)} \tag{5.8}$$

Any state where there is a Commitment to bring about a proposition p*, and at the same time* p *is true are automatically transformed to a state where the condition* p *still exists, but the Commitment has been canceled and does not exist any more.*

In Singh's proposal (Singh 1999), there are two more operations on Commitments that were included. In our model they were integrated as action as part of our model. These are *delegation* and *assignment* of Commitments. These will not be defined extra, but instead two examples of how to represent these operations in the model are presented:

- Delegation: the action d, labeled *delegates*, changes the debtor of a Commitment C from agent a_{d1} to a_{d2}:
  $d = \langle s(C(a_c, a_{d1}, p, c, t)), \text{“delegates”}, \{-C(a_c, a_{d1}, p, c, t), +C(a_c, a_{d2}, p, c, t)\}\rangle$

- Assignment: the action a, labeled *assigns*, changes the creditor of a Commitment C from agent a_{c1} to a_{c2}:
  $a = \langle s(C(a_{c1}, a_d, p, c)),\ \text{“assigns”}, \{-C(a_{c1}, a_d, p, c, t), +C(a_{c2}, a_d, p, c, t)\}\rangle$

5.4.3 Conditional propositions

In order to represent situations with two optional outcomes, depending on a proposition in the conversation, a special kind of proposition name is provided, called *conditional proposition*. It has a proposition as argument (the *condition*) and two sets of operations. Depending on the truth value of the condition, one of two sets of operations are applied.

These kind of proposition names are intended to be used only in operations, and allow the Speech Acts having these operations to have two possible State Descriptions as outcome.

It is important to remark, that in each case, the corresponding proposition (true or false) of the condition, is introduced to the resulting State Description.

5.4.4 Cross-Conversational Constraints

Conversations are performed in parallel, as independent instances (See Figure 5.1). Still, there are cases where a protocol specifies a constraint that refers to other conversations that are being performed as part of a same protocol instance. For example, in the contract-net protocol, the *accept* message should be delivered after all proposals in each conversation are received. This is a constraint for the *accept* Speech Act: that it can only be performed if all other conversations have reached the status specified.

For this purpose, a Cross-Conversational Constraint is defined. It specifies that a proposition should be valid in all conversations in order to perform some action. It is used as precondition for Speech Acts and specifies what proposition should be valid. This kind of constraint sets a break in the parallel execution of conversations and is used to join them in a State Description where some decision can be made by one of the roles, taking into account the information of all the conversations, hence the name “Cross-Conversational”.

Definition 10. *A Cross-Conversational constraint $W([\neg]p, n)$, is a proposition that states that the proposition $[\neg]p$ must have the specified truth value in all conversations for W to be valid. The argument n is the fraction of all*

the current conversations that must have p with value t. If omitted, n has the amount of current conversations, meaning that all conversations must have that value in order to proceed.

The effect such a constraint has on a participant is that it stops it from proceeding until all conversations with partners have reached a certain State Description. The participant will be obliged to **w**ait (therefore the name W) for the conversations to reach that state.

5.5 Definition of a conversation protocol

The following section will provide the mechanisms to compose actions in such a way that they describe how complex conversations are performed.

Protocols will be specified similarly as in (Eijk, Boer, Hoek & Meyer 2003), in terms of the propositions required to start it, called preconditions and propositions describing the effects it has in the context of the conversation, called post-conditions.

Definition 11. *A protocol π is a labeled construct composed of starting State Descriptions, ending State Descriptions and a set of Speech Acts. Starting and ending State Descriptions have associated to cardinality constraints:*

$$\pi : \langle label, \{{}^{w_a}_{v_a}s_a, \ldots, {}^{w_b}_{v_b}s_b\}, \{{}^{w_c}_{v_c}e_c, \ldots {}^{w_d}_{v_d}e_d\}, \{{}^{q_e}_{p_e}sa_e, \ldots, {}^{q_f}_{p_f}sa_f\}\rangle \quad (5.9)$$

where

${}^{w_a}_{v_a}s_a, \ldots, {}^{w_b}_{v_b}s_b$ are the starting State Descriptions with their corresponding cardinality constraints.

$\{{}^{w_c}_{v_c}e_c, \ldots {}^{w_d}_{v_d}e_d\}$ are the ending State Descriptions with their corresponding cardinality constraints.

${}^{q_e}_{p_e}sa_e, \ldots, {}^{q_f}_{p_f}sa_f$ are the Speech Acts that compose the protocol with their corresponding cardinality constraints.

All Speech Acts ${}^{q_y}_{p_y}sa_y$ in a protocol sharing a starting State State Description ${}^{q_x}_{p_x}s_x$ as precondition must satisfy the cardinality constraints associated to the State Description s_x: p_x and q_x :

$$s_x - enables \rightarrow sa_y \Rightarrow p_y \leq v_x \wedge q_y \geq w_x \quad (5.10)$$

All Speech Acts ${}^{q_y}_{p_y}sa_y$ in a protocol that result in a State Description that matches an ending State Description ${}^{w_x}_{v_x}e_x$ of the protocol, must satisfy the cardinality constraints associated to the end State Description e_x: p_x and q_x:

$$s_x - enabledBy \rightarrow sa_y \Rightarrow p_y \geq v_x \wedge q_y \leq w_x \quad (5.11)$$

The graph representation of the protocol can always be calculated from the Speech Acts by finding all possible sequences of chaining them. The result is a directed graph with State Descriptions as nodes and Speech Acts as arcs with possibly many starting and ending nodes. It is important to remark, that protocols that have the same starting and ending State Descriptions are not necessarily the same, but are expected to fit into the same place in a protocol composition.

Actions of a protocol are also called *rules* as used in dialogue games (McBurney & Parsons 2003), where these serve as rules on how each participant can move next.

The more relaxed approach used in our work, facilitates various composition mechanisms for protocols and reflects the multi-directional nature of conversations better.

The complex nature of conversations makes the task of modeling and structuring them very hard. It is the intention of this work to provide mechanisms to organize and modularize complex conversations without restricting them unnecessarily or in such a manner that ends up being unnatural for practical purposes. Therefore a technique for composing protocols using rigid structures that do not always fit the nature of conversations will not be pursued here. Such structures found commonly in similar approaches are probably inherited from programming structures, like *if... then... else...*, *while loops* and specially strict *joining* associated with a previous *split* in the transition system. Our approach allows such structures, but it is by far not restricted to them.

5.6 Protocol composition

Protocol composition is the creation of new conversation protocols by connecting other protocols together.

Definition 12. *Two protocols π_1 and π_2 can be composed to form a new protocol, if there is at least one ending state description s_1 in π_1 that is a subset of a starting state s_2 in π_2 and at the same time, cardinality constraints in s_1 are equal or more restrictive than in s_2.*

Propositions and roles have to be bound *together to establish the semantic connection between the two protocols π_1 and π_2, by specifying which roles and proposition names in the first protocol will take the roles and replace the proposition names in the second protocols respectively:*

$$\pi_1 = \langle''\pi_1'', S_1, E_1, A_1\rangle$$
$$\overset{cq_1}{cp_1} e_1 \in E_1$$
$$\pi_2 = \langle''\pi_2'', S_2, E_2, A_2\rangle$$
$$\overset{cq_2}{cp_2} s_2 \in S_2$$

pi_1 and π_2 can be bound by connecting ${}^{cq_1}_{cp_1}e_1$ and ${}^{cq_2}_{cp_2}s_2$ if:

$cp_1 \geq cp_2$
$cq_1 \leq cq_2$

a specific binding of roles in the form of a mapping $br(r)$ is specified:

$br(r) = r'$

where
r: a role in π_1
r': a role in π_2

a specific binding of proposition names in the form of a mapping $bp(p)$ is specified:

$bp(p) = p'$

where
p: a proposition name in π_1
p': a proposition name in π_2

and
$bp(e_1) \subseteq s_2$
where applying $bp(x)$ to a State Description x applies the mapping to each proposition name inside the State Description.

It is part of the nature of conversations, that subjects as well as participants come and go. Not all proposition names and roles of the first protocol have to be bound to the second one, only those necessary for the connection (the connected starting and ending State Descriptions). The connected protocols may have proposition names and roles not bound to the other one. In the case of the first protocol, these unbound concepts are irrelevant for the continuing conversation. In the case of the second protocol, these concepts are new roles and proposition names that are introduced to the conversation.

5.7 Protocol example

A simple protocol definition and composition example is now provided to show the presented model at work. Subsequent sections in this document will elaborate on this meta-model in depth, thereafter, more complex examples will follow.

In this case, we will look at a simple *order* protocol and a *cash payment* protocol that will be bound together.

The *order* protocol consists of a *requester* that orders (*order*) something from a *provider*. The *provider* can reply either with a *deliver* or a

cancelOrder. In a second protocol: the *payment* for the delivery, will be performed by a *customer* towards a *seller* which will be bound to *requester* and *provider* of the first protocol. The protocols will be bound through an ending State Description of the *order* protocol, where the item has been *delivered*. Therefore the *delivered* property name will be bound to the *paymenPending* in the second protocol.

The objective of this chapter is to establish an unambiguous theoretical foundation. Implementation and practical usage scenarios are presented in the next chapters. This same example is used in the first examples in further sections (see Fig. 6.8), where diagrams are shown. Also, due to the intricate syntax and complexity of more in depth examples, further discussion and description of details are presented in further chapters, where automatically generated diagrams are provided.

$$
\begin{aligned}
&\pi_1 = \\
\Big\langle\ & \text{“}\texttt{order}\text{”}, \\
& \{\{\ \neg C(delivered, provider, requester, \\
& \quad T(t_1, \text{“}\texttt{cancel Order}\text{”})), \neg orderPosted\ \}\}, \\
& \{\{\ delivered, \neg C(delivered, provider, requester, \\
& \quad T(t_1, \text{“}\texttt{cancel Order}\text{”})), orderPosted, \\
& \quad \neg T(t_1, \text{“}\texttt{cancel Order}\text{”}))\ \}, \\
& \{\ \neg delivered, \neg C(delivered, provider, requester, \\
& \quad T(t_1, \text{“}\texttt{cancel Order}\text{”})), orderPosted, \\
& \quad \neg T(t_1, \text{“}\texttt{cancel Order}\text{”}))\ \}\}, \\
& \{\langle\ \text{“}\texttt{order}\text{”}, requester, provider, \\
& \{\ \neg C(delivered, provider, requester, \\
& \quad T(t_1, \text{“}\texttt{cancel Order}\text{”})), \neg orderPosted\ \}, \\
& \{\ -delivered, +C(delivered, provider, requester, \\
& \quad T(t_1, \text{“}\texttt{cancel Order}\text{”})), +orderPosted, \\
& \quad +T(t_1, \text{“}\texttt{cancel Order}\text{”})\ \}\rangle, \\
& \langle\ \text{“}\texttt{deliver}\text{”}, provider, requester, \\
& \{\ \neg delivered, C(delivered, provider, requester, \\
& \quad T(t_1, \text{“}\texttt{cancel Order}\text{”})), orderPosted, \\
& \quad T(t_1, \text{“}\texttt{cancel Order}\text{”}))\ \}, \\
& \{\ +delivered\ \}\rangle,
\end{aligned}
$$

$$
\begin{array}{ll}
\langle & \text{“}\texttt{cancelOrder}\text{”}, provider, requester, \\
\{ & \neg delivered, C(delivered, provider, requester, \\
 & T(t_1, \text{“}\texttt{cancel Order}\text{”})), orderPosted, \\
 & T(t_1, \text{“}\texttt{cancel Order}\text{”})) \qquad\qquad\qquad\qquad\qquad\qquad \}, \\
\{ & -C(delivered, provider, requester, \\
 & T(t_1, \text{“}\texttt{cancel Order}\text{”})), -T(t_1, \text{“}\texttt{cancel Order}\text{”}) \quad \}\rangle\}
\end{array}
\Big\rangle
$$

$$
\pi_2 =
$$

$$
\Big\langle
\begin{array}{ll}
 & \text{“}\texttt{payment}\text{”}, \\
\{\{ & paymentPending \quad \}\}, \\
\{\{ & \neg paymentPending \quad \}\}, \\
 & \\
\langle & \text{“}\texttt{pay}\text{”}, customer, seller, \\
\{ & paymentPending \qquad\qquad \}, \\
\{ & -paymentPending \qquad\qquad \}\rangle\}
\end{array}
\Big\rangle
$$

Protocols are bound as follows:

$\pi_1 \rightarrow \pi_2$

with the following role binding:
$br(r) = r'$:

r	$\rightarrow$	r'
$provider$	$\rightarrow$	$seller$
$requester$	$\rightarrow$	$customer$

and the following proposition name binding:

$bp(p) = p'$:

p	$\rightarrow$	p'
$delivered$	$\rightarrow$	$paymentPending$

5.8 Summary

Our model of interaction protocols is presented in this chapter, which solves the lack of a comprehensive, formal and consolidating model for interaction protocols and connects the definition of Speech Act and Interaction Protocol, in opposition to FIPA's specifications. It consolidates various aspects contributed by the multi-agent systems community and disambiguates concepts. Protocols play the role of a contract between the participants, specifying the rules to follow to perform a dialog.

Our model is based on classical planning, but seen from a global perspective and not from that of an agent, as in planning. It is a transition system

between State Descriptions composed of propositions and their truth values. Transitions between State Descriptions are achieved using Speech Acts which are defined by a set of preconditions and operations on propositions that define how the State Descriptions are to be modified, if the action is performed.

Our model defines various constructs that are inherent to interaction protocols and that differentiate our model from plain planning: Roles, Commitments, Timeouts, etc. The model is enhanced with Cross-Conversational Constraints, a key concept in interaction protocols: it is used to manage the way a set of parallel conversations that enact a protocol are synchronized. It denotes what proposition has to be valid, for any conversation to continue: the difference with a normal precondition is, that this proposition has to be valid in all parallel conversations, before any conversation can go on.

Protocols are the result of an aggregation of Speech Acts, Roles that perform these actions, and State Descriptions. State Descriptions can be of three kinds: starting, intermediate or ending State Descriptions. Protocols, from an external perspective, can encapsulate their contents and show only their starting and ending State Descriptions. This way, they can be compared and interconnected by matching starting and ending State Descriptions, making our model modular.

Chapter 6

Implementation

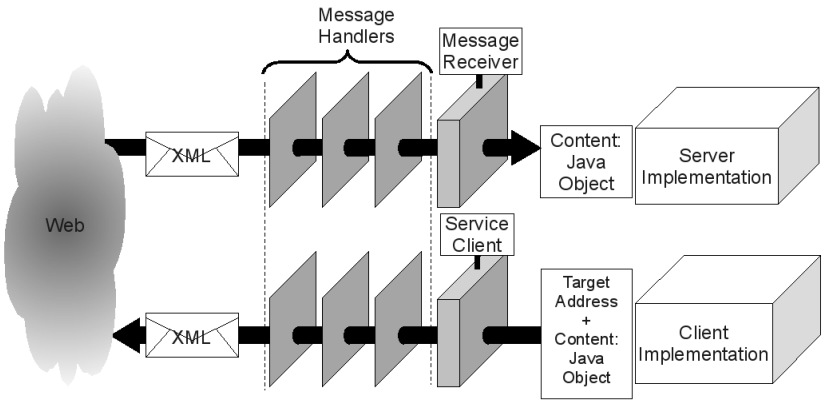
This chapter will provide more insight into how the proposed solutions of Chapters 4 and 5 are implemented. Integration of FIPA and Web Services was proposed as a messaging framework mapping in Section 4.2. Here, implementation details of a tool that uses this mapping will be explained along with details about how some peculiarities are solved, like matching synchronous and asynchronous communication or stateless and stateful entities.

After that, a tool for editing interaction protocols following the proposed model in Chapter 5 will be presented. Details of how the model is programmed, a visual editor and how it supports the designer are presented. At the end a mapping to Jadex agents is explained and a small example shows how the model was used.

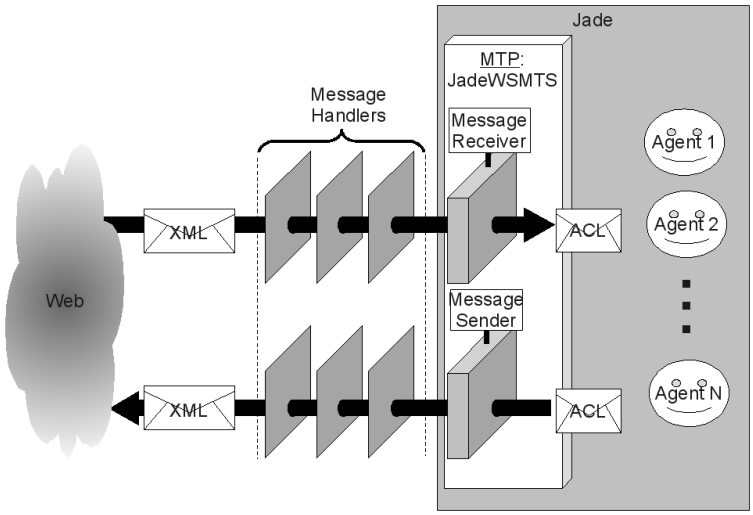
6.1 JadeWSMTS Implementation

This section explains the implementation of a tool called JadeWSMTS (León Soto 2007) for integration of FIPA agents in Jade and Web Services. The Web Services interface will be implemented with a tool called Axis2 (AXIS2 2006), as it is one of the most up-to-date implementations of Web Services standards, particularly WS-Addressing. Using Axis2, consistent maintenance of Web Services standards is delegated, isolating the implementation of JadeWSMTS from changes in Web Services specifications.

Axis2 consists of a set of software components which processes incoming and outgoing messages. Its internal structure is depicted in Figure 6.1a. Each component, called *handler*, does a specific job in processing messages. Each handler is in charge of a specific functionality normally associated to a WS-* standard or a messaging infrastructure function, for instance, finding the *Message Receiver* belonging to the requested service. When an incoming message is processed, XML information in the message is parsed into data structures in Axis2, this data is then provided to an Axis2 Message Receiver



(a) Axis2 Internal structure



(b) JadeWSMTS implementation in conjunction with Axis2

Figure 6.1: Architectural aspect of Axis2 and JadeWSMTS

in order to invoke the corresponding Java *server* code. When using Axis2 to implement traditional Web Services, a Message Receiver is created for each service. Using Axis2 for invoking a service in the Web involves instantiating a Service Client which processes the outgoing message by producing an Axis2 data structure for it. This message information is forwarded to the Axis2 infrastructure (the handlers) for processing, converting it to XML, following WS-* standards and dispatching. Handlers can be added and removed in order to add, remove or customize support for specific Web Services standards or any special functionality.

Architecturally, JadeWSMTS is implemented as an Axis2 Message Receiver and Service Client pair, see Figure 6.1b. Jade provides an extension point that allows registration of different implementations of *Message Trans-*

port Protocols (MTP) in the form of Message Transport Services (MTS). Making use of this feature, JadeWSMTS is registered in Jade as any other MTP and can be used to deliver in the agent platform messages received through Web Services and to dispatch messages sent by agents in the platform to the outside as Web Services messages. Figure 6.1b shows how this implementation makes use of Axis2 handlers taking advantage of all the Web Services related functionality they provide.

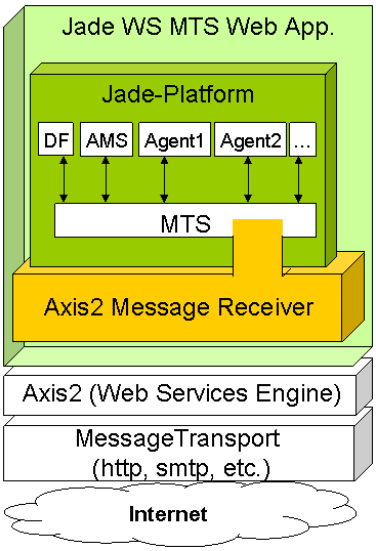
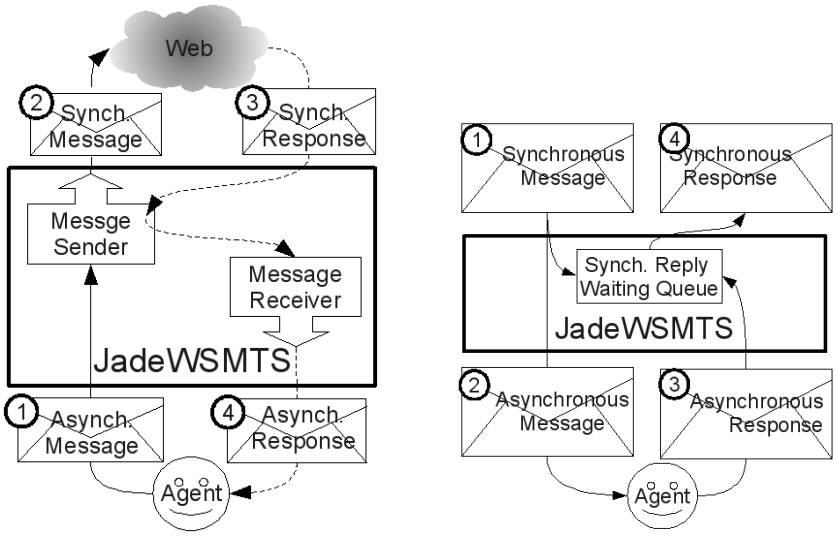


Figure 6.2: Jade WSMTS Architectural stack

An Axis2 Web Application (a service) was created to host a Jade agent platform. Figure 6.2 shows how Jade is integrated into a Web Services environment the same way as any other Axis2 Web Services application. The stack shows how this implementation is based on an Axis2 Web Services Engine and hence, it can run on top of varied transport mechanisms like http, smtp, etc. An Axis2 Web Services applications (JadeWSMTS Web App) contains a message receiver and a Jade agent platform (Jade-Platform). The Message receiver is registered as an MTP inside the agent platform, this is represented as the extension of the message receiver (Axis2 Message Receiver) on its top right that extends into the MTS of the agent platform. This Figure illustrates a reciprocal containment relation that exists between JadeWSMTS and the agent platform: JadeWSMTS contains the agent platform where it delivers incoming messages and at the same time the agent platform contains JadeWSMTS as part of its MTS to deliver messages to the outside.

Agents in agent platforms communicate in an asynchronous way. They send messages using the MTS of the platform, which works similarly to a

postal service. Therefore, agents normally do not stop their execution while waiting for a reply. The most commonly used Web Services, those over http, communicate synchronously. This is evident in the kind of network connection they use: synchronous Web Services over http expect the reply through the same http connection. A client invoking a services connects to it, sends the request and stays on hold with the connection active to receive through it the reply. This difference raises a problem in low level communication.



(a) Synchronous service request

(b) Synchronous incoming request handling

Figure 6.3: Synchronous and asynchronous communication in JadeWSMTS

Normal processing of messages in JadeWSMTS consists of sending messages through the message sender and receiving them through the receiver, in essence it works asynchronously. Even so, JadeWSMTS is capable of making communication of synchronous Web Services and asynchronous agents compatible. This problem arises in two situations: when agents communicate to services that are synchronous and when agents offer services to clients that can only connect synchronously. Figure 6.3a shows how the first situation is handled. Following the numbers surrounded by circles, this figure illustrates an agent sending an asynchronous message (1). Messages can be annotated by the sending agent with the instruction to use synchronous http dispatching (2). This makes the message sender hold the connection and wait for a reply. The reply is received by the message sender (3) and immediately forwarded to the receiver, which handles the incoming message as any other asynchronous message by delivering it to the agent (4). Figure 6.3b illustrates the other case. Here a synchronous message (1) is received by JadeWSMTS and stored in a *synchronous reply* waiting queue. The mes-

sage is also forwarded to the agent (2), which replies asynchronously (3). JadeWSMTS is capable of detecting that the message is the asynchronous response to the message waiting in the queue and sends it through the synchronous connection (4) that was "on hold". The only inconvenience in this case is that the agent attending to this kind of messages should reply fast enough to avoid any connection Timeout problems.

The synchronous-asynchronous feature of JadeWSMTS makes it possible for agents to offer REST services, since they can attend synchronous requests. JadeWSMTS offers, apart from that, another very important feature in order to offer REST services: agents can *register* addresses specific for them. Normally in JadeWSMTS the address used for connecting is that of the agent platform, the specific agent is reached by means of the recipient information (Agent ID) part of the SOAP envelope which states which agent the message is sent to. In REST services, messaging works in a different manner as SOAP messaging. One fundamental difference is that each participant in an interaction should be reachable directly by an address specific for it. Therefore JadeWSMTS offers the possibility to register in the web application another address which will allow the message to reach the agent without the need of extra SOAP envelope headers, in other words: for an address to reach the agent directly. All messages sent to the address of the agent will arrive at this specific agent without the need for any envelope or messaging headers.

6.1.1 Endpoint References (EPR)

The first information type defined in this implementation is the merge of an Endpoint Reference (EPR) (W3C 2006*a*) and a Agent ID (AID) (FIPA 2002*b*).

EPRs, as the name states, are intended to *refer to endpoints*, which make them an object in which a fundamental difference between agents and Web Services becomes evident: Web Services only demand to be *referenced* while agents are to be *identified*. Agents have a stateful nature: when they interact, other participants expect them to take into account what has been done previously in the conversation, in other words to maintain the state of a particular conversation. It cannot be expected from an arbitrary agent that it replaces another agent in the middle of a conversation without updating it with what has happened up to that moment. On the other hand, Web Services are by nature stateless: which specific software entity performs the offered service is irrelevant as long as it fulfills what the published description offers. Therefore, an identity of a service is not necessary, only a reference to it. It must be possible to replace software entities behind a service without service consumers noticing it.

To get around this difference, the extensibility feature of EPRs will be used to enrich it with the information items that compose an AID. Figure

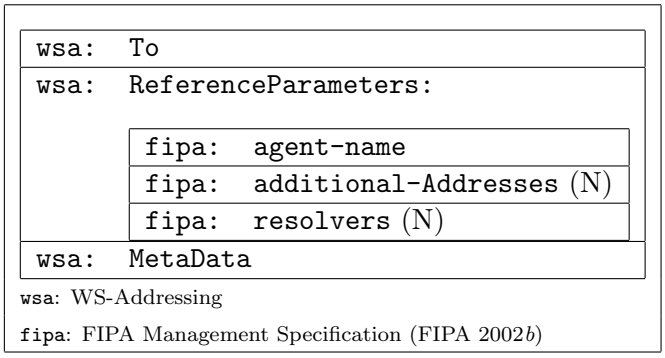


Figure 6.4: FIPA AID–WS-Addressing Endpoint Reference

6.4 shows how an AID has been implemented using a WS-Addressing EPR. The destination address messaging property, which has matching semantics in both specifications, stays as specified for EPRs (`wsa:To`) and will be used as the first address of the agent. The rest of the properties will be added as Reference Parameters: `fipa:agent-name`, the unique identification of the agent, `fipa:additional-Addresses` to store any additional address the agent has and a list of `fipa:resolvers`. Doing so makes, on one hand, FIPA properties transparent for entities that cannot process them. Conventional Web Service consumers, which need only to know how to reach the service using `wsa:to`, will be able to reach agents this way. On the other hand, EPRs from conventional Web Services (which will not have these extra `fipa:` properties in the EPR, particularly `agent-name`) are treated as AIDs of agents that prefer to stay *anonymous*. This implies that agents that will work with conventional Web Services need to be prepared to interact with anonymous entities, which is, in any case, a basic requirement (not only for agents) for interacting with stateless Web Services.

The combination of synchronous and asynchronous communication, interaction based on SOAP-Messaging or REST and the possibility to interact with, or as an anonymous Web Services makes JadeWSMTS a very versatile tool for very different scenarios involving Web Services and agents.

6.1.2 Messaging

The primary contribution of this implementation is the specification of a message envelope for SOAP that merges both properties from FIPA and WS-Addressing standards (León Soto 2006). The envelope follows a structure described in Figure 6.5. The structure of the envelope shows how the envelope specification of WS-Addressing is augmented with FIPA specific properties (FIPA 2002*c*) to support FIPA standardized communication. The four messaging properties written in *slanted* letters are containers for extra

Endpoint References, since the FIPA envelope specification gives these properties plural cardinality in opposition to WS-Addressing, which gives them singular cardinality. The value in `wsa:To` will be used as default target address for the message, the extra headers are expected to be processed only by FIPA entities. Some properties belonging to WS-Addressing have taken the roles of analogous properties in FIPA: action, messageID, inReplyTo and message body. Apart from these, other FIPA messaging properties were added, related to the description of the message contents: encoding, language, ontology, protocol, date and details about payload. Section 4.2 has described in deeper detail how this mapping is designed.

In this tool, agents are free to decide which mechanism to use for a message by making use of `aclRepresentation` envelope property. It can be left empty and the default messaging in Jade will be used or the value identifying SOAP representation can be set, in this case Jade will automatically choose JadeWSMTS to deliver SOAP messages since it is regiestered as the MTS for handling this kind of `aclRepresentation`.

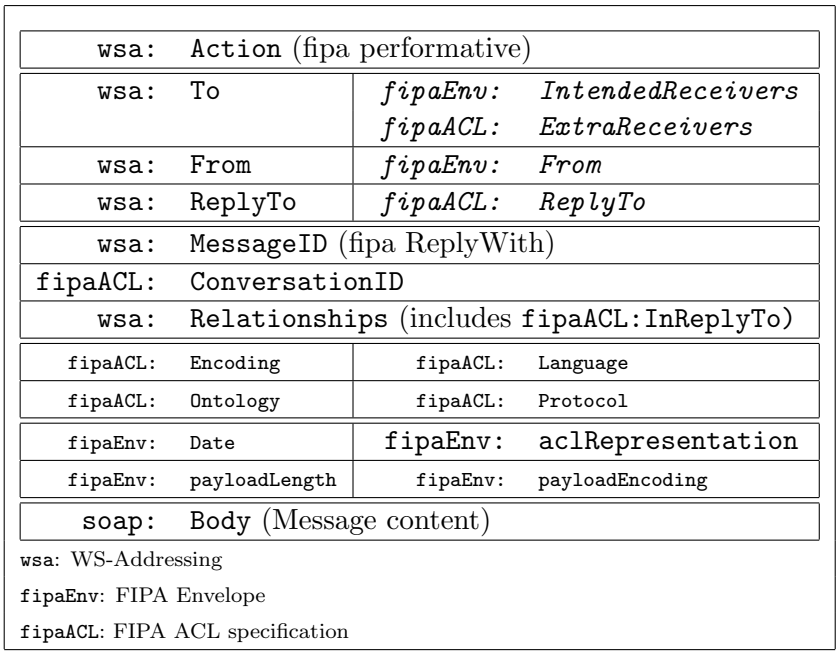


Figure 6.5: FIPA–WS-Addressing message envelope

Message Transport

For some applications a different networking transport might be preferred. In the case of JadeWSMTS, message transportation is delegated to Axis2 whenever SOAP messages go outside the agent platform. Axis2 also allows

the use of other transport protocols, as Figure 6.2 shows, implementations for http and smtp are already provided, and others can be implemented. (Palanca, Escrivá, Aranda, García-Fornes, Julian & Botti 2006) is an example of an implementation on top of another transport, in this case XMPP (IETF 2004), an instant messaging transport protocol. Such an implementation would be even better and profit better of JadeWSMTS and Jade if an implementation for Axis2 is provided. Axis2, JadeWSMTS and Jade would be able to run on top of it and a more standardized and reusable tool would be achieved with less work and more impact, because standards would be used in a wider and more transparent way. From this perspective, JadeWSMTS is also a good tool for extending communication capabilities for JADE.

Message Contents:

Web Services and FIPA standards leave the definition of a content metamodel open. Even so, it is important to remark the dominance of XML for content representation, a tendency well supported by Web Services and partially adopted by FIPA standards. FIPA also provides a *Semantic Language* (SL) (FIPA 2002*h*) for the representation of contents. This being the most frequently used representation for messages that refer to FIPA ontologies. It was also convenient to provide SL representation in XML. A serialization codec, called FIPA-XML-SL, allows other participants that do not support traditional FIPA String representation, to interact with agents that use SL as a grounding for their contents. It produces contents based on the schema specification provided in (DFKI 2007). This schema can be used for content type definition in a WSDL description of an agent.

6.1.3 Publication and discovery

From the perspective of messaging, the concepts of publication and discovery are a specific kind of content specification (León Soto 2006). Therefore, as already stated in section 1.3, publication and discovery are not part of the integration provided by JadeWSMTS.

Even so, it is important to remark that this tool can be used for publication and discovery using the already existing tools in FIPA or Web Services. FIPA architectures provide two registry services (FIPA 2002*b*), the *Agent Management Service (AMS)*, and the *Directory Facilitator (DF)*. Publication and discovery, are performed by interacting with these services using FIPA SL language. It is possible to provide Web Services interaction with these services using FIPA-XML-SL, presented in Section 6.1.2. This way FIPA registries can also be used as Web Services repositories.

Agents in Jade are also capable of using Web-Services-based registries outside the agent platform, for instance: Universal Description, Discovery

and Integration (UDDI) (OASIS n.d.). An integration of the DF and UDDI concepts will not be approached in this implementation since these are considered different solutions for similar problems as already stated in Section 1.3. Both possibilities are enabled as well as any other facility accessible through Web Services like, for instance, semantic matchmakers.

6.1.4 Message Example

The example in Listing 6.1 shows a message sent using JadeWSMTS. This message is sent by agent `TestAgent1` of `Jade-WebServices-Platform1` (lines 4-13) to agent `df` (the directory facilitator) of `Jade-WebServices-Platform2` (lines 14-19). Some message annotations are added (lines 20 - 30). Note the value for the `acl-representation` (line 26), which identifies the codec used by Jade for processing the envelope. The last header for the message is the action identifying which speech act is being performed (lines 28-30). Then comes the message contents, as mentioned in line 22, it is represented using FIPA-XML-SL language presented in section 6.1.2. As stated in line 29, it is a `request` described using `FIPA-Agent-Management` ontology (line 23) for the `actor` (lines 34-45) to perform the action (lines 46-60) of `registering` (line 46) the agent description of the agent sending the message (lines 47-58).

Listing 6.1: WS-FIPA message example

```
1  <soapenv:Envelope xmlsl:soapenv="..." xmlsl:wsa="..."
2    xmlsl:fipaEnv="..." xmlsl:am="..." xmlsl:acl="...">
3  <soapenv:Header>
4    <wsa:From>
5      <wsa:Address>
6        http://localhost:8085/axis2/services/MTS
7      </wsa:Address>
8      <wsa:ReferenceParameters>
9        <axis2ns4:agent-name>
10         TestAgent1@Jade-WebServices-Platform1
11       </axis2ns4:agent-name>
12     </wsa:ReferenceParameters>
13   </wsa:From>
14   <wsa:To>
15     http://localhost:8195/axis2/services/MTS
16   </wsa:To>
17   <axis2ns3:agent-name wsa:IsReferenceParameter="true">
18     df@Jade-WebServices-Platform2
19   </axis2ns3:agent-name>
20   <wsa:MessageID>12356671570200906-0</wsa:MessageID>
21   <acl:conversationID>11176570200906</acl:conversationID>
22   <acl:language>fipa-xml-sl</acl:language>
23   <acl:ontology>FIPA-Agent-Management</acl:ontology>
24   <acl:protocol>fipa-request</acl:protocol>
25   <fipaEnv:acl-representation>
26     fipa.acl.rep.soap.dfki.v.0.1
27   </fipaEnv:acl-representation>
28   <wsa:Action>
29     http://dfki.de/fipa/speechacts/request
30   </wsa:Action>
31 </soapenv:Header>
32 <soapenv:Body>
```

```
33 <sl:action-expression xmlsl:ns="...">
34   <sl:actor functionSymbol="agent-identifier">
35     <sl:parameter name="name">
36       <sl:value> <sl:stringValue>
37         df@Jade-WebServices-Platform2
38       </sl:stringValue> </sl:value>
39     </sl:parameter>
40     <sl:parameter name="addresses">
41       <sl:value>. . .
42           http://localhost:8085/axis2/services/MTS
43         . . .</sl:value>
44     </sl:parameter>
45   </sl:actor>
46   <sl:action functionSymbol="register">
47     <sl:operand functionSymbol="df-agent-description">
48       <sl:parameter name="name">
49         <sl:value functionSymbol="agent-identifier">
50           . . .
51         </sl:value>
52       </sl:parameter>
53       <sl:parameter name="protocol">
54         <sl:value> <sl:element>
55           <sl:stringValue>fipa-request</sl:stringValue>
56         </sl:element> </sl:value>
57       </sl:parameter>
58     </sl:operand>
59   </sl:action>
60 </sl:action-expression>
61 </soapenv:Body>
62 </soapenv:Envelope>
```

6.2 Declarative Protocols Implementation

The intention of having a meta-model like the one defined in Chapter 5 is to produce tools that support the development of models and are capable of producing executable code. The current section will describe how these tools were implemented. First a meta-model implementation was created, than a diagram editor was implemented to design models visually and finally a mapping from our models to runnable Jadex agents code was programmed (León Soto 2012).

The Eclipse Modeling Framework (EMF) is going to be used for the implementation of these tools. It is the same framework used to implement PIM4Agents (Section 3.5.13) and MDA tools related to this project.

6.2.1 Implementation of meta-model using EMF

The first step in creating an MDA-tool is to define the meta-model to be used. A summary of the meta-model for Declarative Protocols is presented in Figure 6.6, a detailed description of the meta-model has already been semantically specified in depth in Chapter 5.

The core concept in Figure 6.6 is *DeclarativeProtocol*, which is defined by *ACLMessages* (Speech Act). This is a relation that is inherited from the

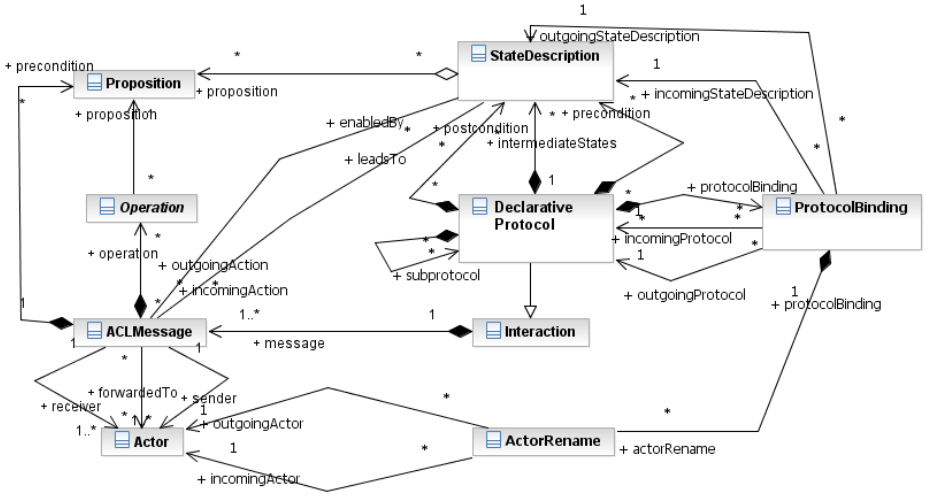


Figure 6.6: Key Concepts of Declarative Protocols.

concept of *Interaction* in PIM4Agents. At the same time, it contains *Actors* (in the figure it is shown through ACLMessage) and State Descriptions.

State Descriptions are contained in protocols in three different ways, either as starting State Description (precondition), intermediate State Description or ending State Description (post-condition).

ACLMessages, apart from containing Actors as sender and receiver, are defined by a set of *Operations* and a set of *Propositions*. These propositions are their pre-conditions. The operations of an ACLMessage are also defined based on proposition names. ACLMessages are connected to State Descriptions that can be either State Descriptions the message is *enabledBy* or the message *leadsTo*.

On the right side of the figure are the concepts related to *Protocol Binding*. A Protocol Binding is the concept used to connect two protocols to produce a composed protocol. It connects an ending State Description of a protocol as incoming State Description to a starting State Description of another as outgoing State Description. It also contains a mapping of Actors called *ActorRenames* which establish for roles in the first sub-protocol what role to take in the next sub-protocols.

Even though the concept of State Description is defined in this model at the same level as ACLMessage and Proposition, it is not intended to be a concept to be introduced by the user, nor its associations to ACLMessages or Declarative Protocol. Those are concepts that will be automatically calculated out of the ACLMessage definitions. Protocol designers only need to define roles, propositions and ACLMessages and the State Description

structure will be completed for them by the implemented meta-model.

6.2.2 Editor for Declarative Protocols

One of the most relevant reasons why a declarative approach has not been supported by the MAS community is that it is inevitably more complex than current approaches. The amount of detail that such an approach brings along tend to make models less intuitive for human readers(Miller & McBurney 2007). This is specially the case for language-based modelling tools, the most common approach for modeling in computer science.

We are interested in producing executable code from the specified models. The implementation of an MDA-based modeling tool aims to have these two aspects:

- Cope with complexity: using graphical diagramming tools, complexity of models can be controlled with ease. Editors can be enhanced with supportive logic that makes management of information in the models less effort demanding. Apart from that, diagrams are very helpful in exposing models in a more intuitive manner.

- Code generation: MDA techniques target this aspect. Models can be used as input for transformation tools that produce Platform Specific Models (PSM) which represent how a model is implemented in a particular platform. From PSMs, executable code is produced. The mapping of Declarative Protocols to executable code will be explained in Section 6.2.3.

A visual editor for Declarative Protocols has been implemented using MDA tools based on the eclipse modelling framework (EMF), like the Graphical Modeling Framework (GMF).

The intention of the graphical editor is to allow users to add to a protocol model concepts like roles, proposition names and use these to define Speech Acts with preconditions and operations. The editor maintains, based on these concepts, a directed graph that represents the possible paths a conversation can take following the protocol.

Visual Editor modeling layers

The editor provides two kind of views called:

- **Composition layer**: shows protocols in the form of labeled rectangles which have entry and exit points attached to their perimeter. Figure 6.7 shows as an example, an excerpt of the model explained in section 7.2.4. Entry and exit points represent starting or ending State Descriptions of the protocol. Ending State Descriptions of a protocol

can be connected to starting State Descriptions of other protocols, to establish a transition from one protocol to another. These transitions are known as State Description-bindings and are represented by black arrows.

In order to know how a conversation goes from one protocol to another in a transition, Role-bindings and Proposition Name-bindings are defined (not shown in figure). The first one defines which role in the first protocol is "taken over" by which role in the other protocol. In other words, the agent performing a role in a protocol will be performing the role in the second protocol that is linked by the same Role-Binding. Hence, Role-Bindings are sets of roles in different protocols that will be performed by the same agent.

Proposition Name-Bindings, similarly, define which Proposition names are translated to which Proposition names in the second protocol. They can be seen as sets of synonyms. Based on the information provided by them, if a starting or ending State Description is selected, the editor is capable of highlighting State Descriptions that are compatible and can be connected.

- **Protocol layer**: by selecting a protocol in the composition layer, its contents are shown using a view in the protocol layer. Figure 6.8 illustrates such a view.

  At the top, the declaration of concepts to be used in the definition of the protocol are declared. They can be simple Proposition Names or complex ones like Timeouts, Commitments and Conditional Propositions. Also the roles are defined, these will be used as sender or receivers of messages in the protocol.

  The protocol is developed by defining Speech Acts, shown as rectangles with a white background, by specifying its roles (sender and receiver), cardinality constraints, preconditions (in the middle subsection of the rectangle) and last, but not least, its operations.

  Right after defining Speech Acts or modifying them, State Descriptions, shown as blue rectangles, are calculated or recalculated automatically.

Example of a protocol diagram

The Declarative Protocol editor represents features of the model using graphical elements. Figure 6.8 shows an example of a Declarative Protocol diagram. On the top left, the proposition names used in the protocol are declared. Speech Acts are rectangles with two compartments and State

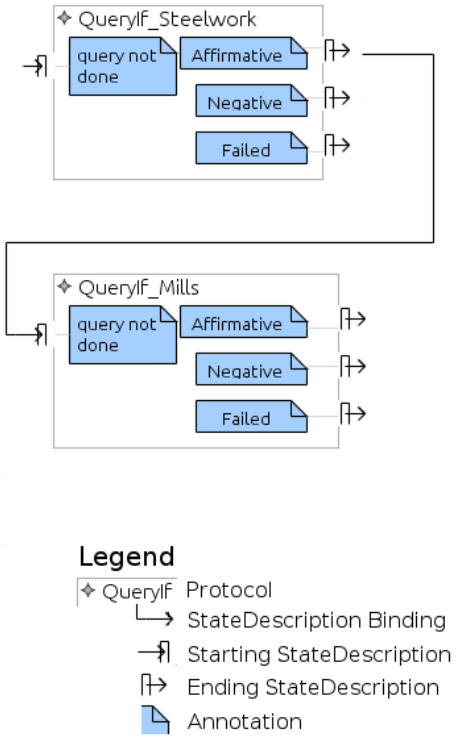


Figure 6.7: Simple Example of the Composition Layer

Descriptions are rectangles with rounded corners. Arrows show the flow between actions and states.

This example is about a small protocol presented in Section 5.7, where two actors, a requester and a provider interact to perform an order. It consists of a Speech Act called *order* from the requester to the provider and two possible responses: *deliver* and *cancelOrder* from the provider to the requester.

Performing *order* creates a Commitment (seen as a gray rectangle on the top) from the provider to the requester to deliver (make delivered true) within 20 time units, otherwise *cancelOrder* will be performed, therefore the corresponding Timeout is also enabled. To perform an *order*, there is a precondition of not having a Commitment established already, to avoid ordering the exact same thing, if it has already been ordered.

Having the Commitment and the Timeout established are preconditions for the next two actions. The Speech Act *deliver* makes delivered true, which automatically disables the Commitment (since it has been honored) and the associated Timeout. This is reflected in the derived State Description: delivered true and no pending Commitments or Timeouts with orderPosted true. The Speech Act *cancelOrder* disables the Timeout (since the Speech

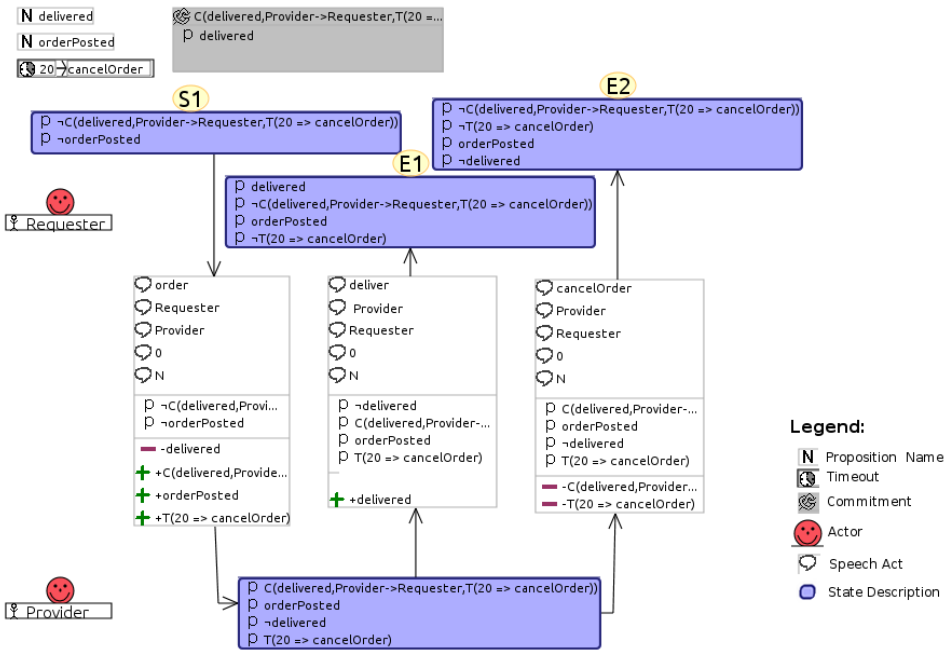


Figure 6.8: Declarative Protocol Diagram example: the "Order" Protocol

Act is the consequence of it) and also disables the Commitment, leading to a State Description where it is free of the Commitment and delivered is still false.

Note that in this example, dishonoring the Commitment, by canceling the order has no negative consequences for the Provider actor. If *cancelOrder* does not disable the Commitment, a State Description will come as result, where the Commitment is valid, the Timeout is over and the objective is still not fulfilled, representing clearly a situation of a dishonored Commitment.

When the user inserts content elements into the model diagram, other graphical details are added or updated automatically, in order to express the semantics and effects of the added element. It uses an algorithm that calculates the changes in a graph that represent the protocol as specified by its actions. Every time the user edits a Speech Act in the protocol, the editor changes the State Descriptions this Speech Act is related to and propagates the changes throughout the graph.

The approach of updating the graph, starting from the area around the modified Speech Act, is preferred to a complete recalculation of the whole graph after each edition, not only because of the potential advantage of performing less calculations, but it also avoids changing the graphical information of unrelated areas, an important advantage for a visual editor of this kind.

Graph updating algorithm

The algorithm for updating the diagram is intended to react to each user edition by adding or updating graphical details that illustrate what has been done. It uses a simple technique similar to forward planning, but differentiating itself in two basic aspects:

- It does not produce single plans, but the graph that represent all possible plans that can be done using the defined actions.
- It does not reflect any intention or particular perspective. Instead, it combines actions that will be performed by different agents and that are defined from a global perspective to calculate the possible states (State Descriptions) and how actions move conversations from one state to another.

For updating the diagram, the algorithm is designed to change only the part of the model that is connected to the concept that has been edited. This process is done recursively in case the change or its effects require updating subsequent parts of the diagram. In other words, the algorithm is not intended to produce a complete graph out of a set of actions, but instead it updates an existing graph after every change that is done to the model.

The main idea behind the calculating algorithm is to disconnect the updated Speech Act completely and then reconnect it correspondingly. First, it is connected to State Descriptions already found in the protocol that meet its preconditions. A new State Descriptions is created if the specific situation that enables the action is not found. Once all possible causing situations are linked, the corresponding effects are calculated.

In case a State Description matching the resulting effects is found, it is connected to the Speech Act as enabled State Description, if not the missing State Descriptions is created and added. In case a new State Description is added, all other Speech Acts are checked against it, to establish possible enabling associations, for which, a resulting State Description is calculated and also checked against all Speech Acts. In case the result is not present in the graph, the process is repeated with it in a recursive manner.

The operation will be explained using pseudo-code. The next code section shows the entry point for the calculations. Every time a speech act represented by *sa* is edited (added, changed or removed), it is first disconnected from its enabling State Descriptions (Line 2). The recursive calculating process is started by calling the *connectEnabling* procedure giving *sa* as parameter (Line 3). Once the process is done, a model cleanup is performed, removing all State Descriptions that are left disconnected:

The procedure *connectEnabling* is shown in the following code segment. It first disconnects *sa* from its enabled State Descriptions and eliminates those that are left without an enabling Speech Act (Lines 2-9). Then, all

```
Starting point of the algorithm when a Speech Act (sa) has been changed:
(1) begin
(2)   sa.enabledBySD = null;
(3)   sp := sa.getPreconditionsAsSD();
(4)   call connectEnabling(sa, sp);
(5)   cleanup(protocol);
(6) end
```

State Descriptions already existing in the model that *match* the precondition (represented by *sp*) are added as an enabling State Description to *sa*. A State Description matches another one, if the states it represents are a subset or equal to the second one (Line 13). Only State Descriptions on which *sa* is *effective* (where performing *sa* produces a change) are considered (Line 14). The State Description is set as enabled by *sa* (Line 20), if it has been caused by another action (Line 15) or it matches exactly the preconditions of the action (Line18). In case there is not an exact match, *sp* is added to the diagram (Line 24-29).

Finally, for each enabling State Description in *sa*, its result, called *sn*, is calculated (Lines 31-32). Results that are effective (Line 33) are connected using the function *connectEnabled* (Line 34). The function returns a boolean value telling if *sn* is a new State Description that has to be added to the model. If this is true, then it is added to the list of State Descriptions that are to be tested for connection (*toConnect*) against the complete model(Line 35).

The next code section shows the function *connectEnabled* which handles the connection of new enabled State Descriptions *sa* has.

The function is called; by passing it the Speech Act *sa* and the specific resulting State Description *sn*, to be considered for connection (Line 1). Each State Description (*sd*) in the protocol (Line 4) is compared to *sn* (Line 5) and then if it is also effective on *sa* (Line 6) it will be connected to *sa* as an enabled State Description (Line 8). In case no State Description in the protocol matches *sn* (Line 12) it is added as enabled, added to the protocol and the value *true* is returned (Lines 14-18).

Finally, the last code fragment shows the recursive function that connects new State Descriptions to the rest of the graph.

Procedure *connect* receives the State Description *s* to be evaluated against the whole graph (Line 1). It starts by checking which Speech Acts in the protocol (Line 2) are enabled by and effective on *s* (Line 3-4). If that is the case, then *s* is made an enabler of the Speech Act (Line 7). Its resultant state *sn* (Line 7) is made a State Description enabled by *sa* (Line 8). In case the State Description was not already part of the protocol (Line 12), it is added and a new recursive call is done for *sn* (Line 13). The recursive search stops when no more State Descriptions are created.

Procedure for connecting enabling SDs:

```
(1)  proc connectEnabling(sa, sp) ≡
(2)    foreach ( esd ∈ sa.enabledSD)do      disconnect from enabled SDs
(3)      if esd.enablingSA − sa ≠ ∅
(5)        then destroy(esd)
(6)        else esd.enabledBY− = sa
(8)      fi
(9)    od
(11)   foreach ( sd ∈ protocol)do           connect all SDs that enable sa
(12)     samefound := samefound ∨ sd = sp;
(13)     if sd ⊆ sp
(14)       sa.isEffectiveOn(sd)∧
(15)       (sd.enabledBy ≠ ∅∨
(16)       sd = sp)
(17)       then
(18)             sa.enabling+ = sd;
(19)     fi
(21)   od
(22)   if ¬samefound
(23)     then
(24)           sa.enablingSD+ = sp;
(25)           toConnect+ = sp;     add to the list of SDs to be connected
(26)           protocol.stateDescriptions+ = sp;
(27)   fi
(29)   foreach ( esd ∈ sa.enablingSD)do      Get all new SDs that arise
(30)     sn := sa(esd);
(31)     if sn ≠ esd
(32)       then if connectEnabled(sn, sa)
(33)              then toConnect+ = sn;
(34)            fi
(35)     fi
(36)   od
(38)   foreach ( sd ∈ toConnect)do call connect(sd); .
```

Procedure for connecting enabled SDs:

```
(1)  funct connectEnabled(sa, sn) ≡
(2)    ret := false;
(3)    samefound := false;
(4)    foreach ( sd ∈ protocol)do   connect all SDs that are enabled by sa
(5)      samefound := samefound ∨ sd = sn;
(6)      if sd = sn ∧ sa.isEffectiveOn(sn)
(7)        then
(8)              sa.enabled+ = sd;
(9)      fi
(11)   od
(12)   if ¬samefound
(13)     then
(14)           sa.enabled+ = sn;
(15)           protocol.stateDescriptions+ = sn;
(16)           ret = true;
(17)   fi
(18)   return(ret).
```

Procedure to connect state description 's' to any related Speech Act

```
(1)  proc connect(s) ≡
(2)    foreach sa ∈ protocol.speechActsdo
(3)      sp := sa.getPreconditionsAsSD();
(4)      if s ⊆ sp ∧ sa.isEffectiveOn(s)
(5)        then
(6)              sa.enablingSD+ = s;
(7)              sn := sa(s);
(8)              sa.enabledSD+ = sn;
(9)              if sn ∉ protocol.stateDescriptions
(10)               then
(11)                     protocol.stateDescriptions+ = sn;
(12)                     call connect(sn);
(13)             fi
(14)     fi
(15)   od.
```

Brief analysis of the algorithm

The algorithm is a routine that implements a complete plan search over a state-action space using methods similar to forward planning.

For the purpose of our diagramming tool, a diagram has to show all and only those State Descriptions reachable using the actions specified by the protocol.

The criteria used to consider which State Descriptions should appear is based on the subset relation '$\subseteq$' between State Descriptions. All State Descriptions are sets of concrete states that fulfill the constraints specified by the State Description on propositions (León Soto 2009). A subset of a given State Description is the result of adding extra constrains on the given State Description, therefore it has the same constraints plus some additional ones in its definition. For instance, the statement $sd_1(a, \neg b) \subset sd_2(a)$ is valid, since sd_1 has the same constraints as sd_2, namely a and additionally $\neg b$. Two Sate Descriptions are the same if they have exactly the same propositions.

New State Descriptions are only created in two circumstances: either by the procedure

sa.getPreconditionsAsSD()

which produces a State Description that represents the preconditions in sa or by the procedure

$sa(sd)$

which produces the results of performing the Speech Act sa on the State Description sd.

Links between State Descriptions and Speech acts are only established as follows:

- State Descriptions can only be linked to Speech Acts as *enabling* State Descriptions, if they are a subset of or equal($\subseteq$) to the State Description representing the preconditions of the Speech Act.
- State Descriptions can only be linked to Speech Acts as *enabled* State Description, if they are equal (=) to the resulting State Description obtained by performing the Speech Act on any of its enabling State Descriptions.

The reason for the first one is that a subset of the preconditions of a Speech Act fulfills the preconditions. In the opposite case, the reason for the second one is that super sets of resulting State Descriptions cannot be associated as result since they contain states that are not part of the resulting State Description, and subsets of the resulting State Description cannot be associated since these would leave out states that are part of the resulting State Description.

Also, since some enabling State Descriptions are related to them by the subset relation, it is probable that resulting State Descriptions will reflect this same relation.

The algorithm is therefore not capable of introducing new State Descriptions that do not have a direct Speech Act as an originator or require it as a precondition. It also does not leave State Descriptions in the diagram if an modification by the user demands them to be removed.

Before the process starts, all enabling State Descriptions are disconnected from the edited Speech Act (but still not removed from the model). Also in the procedure *connectEnabling* in lines 2-9 it is also disconnected from enabled State Descriptions, in this case it removes any State Description that is left without a Speech Act that produced it. The algorithm checks again and reconnects only State Descriptions that match any of the two previously explained cases.

The algorithm adds new State Descriptions if there was no State Description found to exactly match the preconditions or result in the State Description of the Speech Act. For this newly added State Description, a recursive check is started. It checks for any connection possibility with any Speech Act in the model. If it finds a Speech Act that can use this new State Description as enabling and that again produces a new State Description as a result of performing the Speech Act on it, a recursion is done using the new resulting State Description as parameter.

At the end, a cleanup of the model is done, removing all disconnected State Descriptions, leaving the model free of State Descriptions that could not find a relation to any Speech Act in any state.

By observing these properties of the algorithm, it is clear that it always produces the State Descriptions required by Speech Acts, that they are linked to all Speech Acts that can be related to them in the model, and that there are no State Descriptions left in the model without having a Speech Act linked to them.

Combining Protocols as modules

The editor provides a view of all the protocol parts of a project in a simplified way. It reduces protocols to boxes where only their name, starting and ending State Descriptions can be seen. The purpose of this is to allow the protocol designer to produce composed protocols by combining the available protocols.

Protocols can be combined by linking the ending State Description of one protocol with the starting State Description of another. By doing this, a designer says that in the composed protocol, conversations that ended in the ending State Description of the first protocol can continue a conversation in the second protocol by starting it in the starting State Description the link is pointing to. Figure 6.9 shows the project that contains the protocol

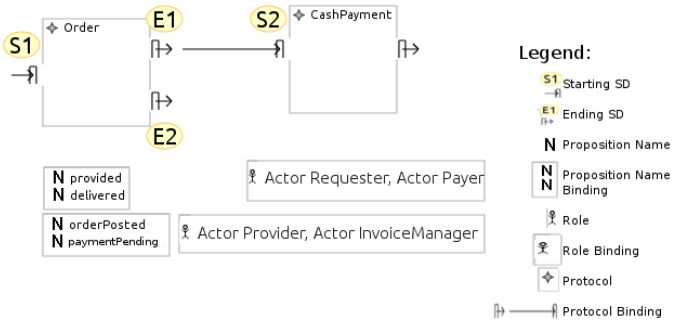


Figure 6.9: Declarative Protocols bound in the Project View.

called "Order" shown in Figure 6.8.

Labels, in the form of ovals have been added in order to help the reader recognize the State Descriptions in the Declarative Protocol Diagram (Figure 6.8) and in the Project View (Figure 6.9): the "Order" protocol has one starting State Description labeled **S1** and two ending State Descriptions labeled **E1** and **E2**. In this diagram, there is only one other protocol defined, called "Cash Payment". Its trivial content is not shown, but it consists of a starting State Description, labeled **S2**, defined by the proposition paymentPending and provided. It has only one Speech Act from a role seller to a role customer called pay which makes paymentPending false and provided true.

Protocols are defined using their own Proposition Names and Actors which form the ontology used to describe the protocol. Since it is not possible to know which State Descriptions match between protocols defined with different Proposition Name sets and Role Names, special binding objects have been added to allow designers to declare which Proposition Names and Roles are to be seen as the same.

These objects are called *Proposition Name Binding* and *Role Binding* and collect concepts from different protocols that can be understood as synonyms. In the example, there is one Proposition Name Bindings on the lower left corner: one saying provided and paymentPending are synonyms. There are two Role Bindings on the lower right: one saying requester and customer which are roles played by the same agent. The other saying provider and seller are also the same.

Taking these bindings into account, one can see the State Description labeled **E1** with the propositions:

E1: (delivered,
¬ C(delivered,Provider→Requester,T(20 ⇒ cancelOrder)),
orderPosted, ¬ T(20 ⇒ cancelOrder) (time: 20)))

is the subset of the State Description labeled **S2**:

S2: (provided, paymentPending)

$$\Rightarrow \mathbf{E1} \subset \mathbf{S2}$$

Therefore, these State Descriptions can be linked and a composed protocol is produced consisting of a first phase where an item is ordered and a second phase where the item is payed.

For simplicity, the example shows only two protocols that are being bound. The objective of this view is to have several protocols available. For instance, in this case more payment methods and not only cash payment. If the Proposition Names and Roles are well bound, more payment protocols can be added to the composed protocol, as modules, simply by linking their starting State Description to **E1**. The result would be a composed protocol with an ordering phase and several payment methods.

Moreover, this view has a feature implemented that highlights matching State Descriptions of other protocols, when an ending or starting State Description is selected. This helps the designer find possible candidates for composition of protocols.

6.2.3 Mapping Declarative Protocols to Jadex BDI Agents

The model produced using the diagramming tool presented in the previous section can be used as input to create an implementation capable of performing the protocol.

In our case, we are interested in creating Jadex-BDI agents (Pokahr, Braubach & Lamersdorf 2005*b*) that perform conversations as specified by the interaction protocol. The Declarative Protocol models will be mapped to Jadex models using a Query View Transformation (OMG 2008), a procedural language for building EMF models, with information from of an EMF source model.

For each role in the protocol, a Jadex *capability* will be produced, where the parts of the protocol related to the role will be implemented in the form of plans. Since each Speech Act in the protocol has a sender and a receiver role, plans for sending and receiving the message will be created in the corresponding capabilities.

Conversations are started by an agent, called the *initiator* of the conversation. This agent addresses a set of participants within the context of a protocol. The initiator agent creates a *conversation context* for each conversation with a participant. Agents ready to participate in conversations, called *participants*, also create a conversation context every time they are invoked to start a new conversation.

Conversation contexts are data objects that aggregate details about the specific conversation between the agent and its partner, in other words, they represent the situation the conversation is in. In the Declarative Protocol model, this situation is represented by the State Descriptions. The values of the propositions as specified in the State Descriptions are used in the mapped

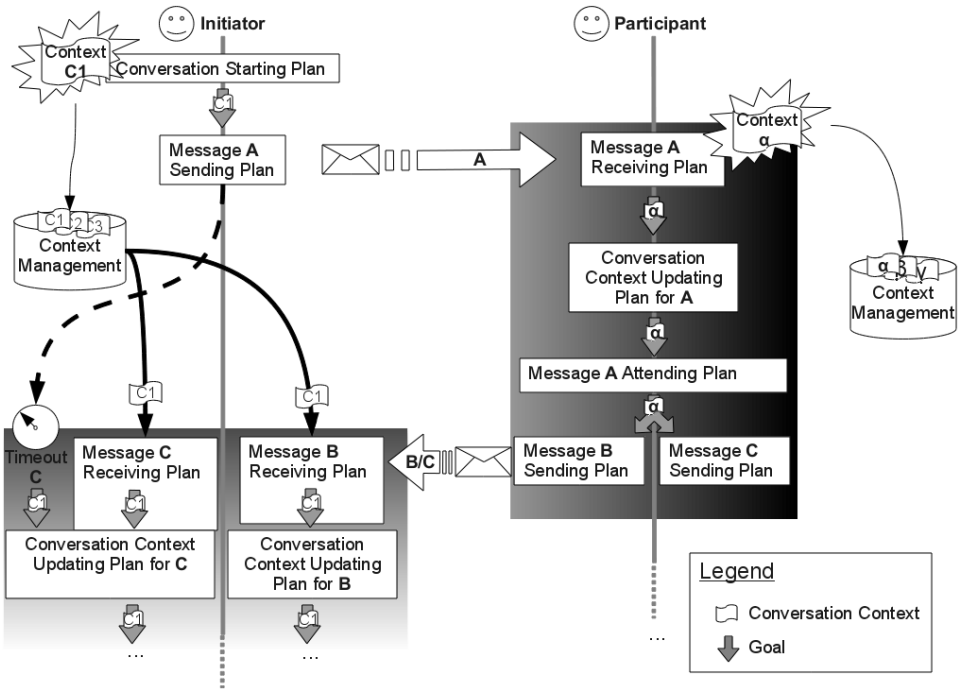


Figure 6.10: The BDI-Plans pattern for Declarative Protocols used by Participants performing Conversation Protocols.

capability to update the conversation context as the dialogue is carried out. Conversation contexts are the representation of State Descriptions inside the beliefs of participating agents.

The process of receiving a message, reasoning about it and producing a reply is called a *turn*. A pattern has been specified for implementing turns in a capability and it consists of a set of plans that handle its different stages.

The pattern used for implementing turns is shown in Figure 6.10 and how conversation contexts are created, managed and passed from plan to plan. It shows the beginning of a conversation in a canonical way: protocol enactment between several parties can be decomposed into "dialogs" between two parties, one is an initiator and another one is the participant. The figure shows, looking from top to bottom, the sequence in which the plans are performed.

The first detail to observe is where conversation contexts are created, which is symbolized by a punched tape symbol (a sinuous rectangle) inside a star. For the initiator, the context is created when it decides to create the conversation with the partner. It will do the same for each partner it interacts with. The Participant, since it plays the passive role in the conversation, creates a context every time it receives the first message specified in a protocol. Not all message receiving plans create conversation contexts,

only those that receive the first message sent to a participant as specified in the protocol. Therefore, during the mapping process, message receiving plans are annotated with a information expressing whether they can create new conversation contexts, like message receiving plan for message **A**. Those that do not have the annotation will instead fetch the corresponding context from the context management when a message is received. In the example, this is the case for the receiving plans for messages **B** and **C**.

The dark rectangles enclose the basic BDI-Plan pattern used for implementing turns in a dialog. For instance, the rectangle on the right shows the first turn taken by the participant. Messages are received and the corresponding context is created or fetched. Then, a goal for updating the conversation context based on the message that arrived is posted. The corresponding "Conversation Context Updating Plan" updates proposition values in the context, as it was expected, since the message was received. By extending this plan, the agent designer can improve the updating process and context checking by introducing analysis of details in the content of the message.

After that, a goal for actually attending the message is posted which triggers the corresponding "Message Attending Plan". This is the plan in which the reasoning and internal processing for that turn is programmed. Depending on what message is intended to be sent as a reply, the agent chooses what goal to post. A corresponding "Message Sending Plan" is triggered and performs the message sending and the process starts again on the side of the recipient agent. In our example, there are two possible replies for message **A**: **B** and **C**. For each of these messaging events, there is a pattern on the side of the recipient, in this case the initiator, ready to first receive the message, then to fetch the corresponding conversation context stored in the context management. From here on the pattern is repeated throughout the conversation.

The conversation context is passed on from plan to plan as a goal parameter, until a message sending plan is performed ending the turn and leading the course of the conversation to the agent the message is addressed to. The example shows how each agent manages its own conversation contexts: the initiator calls the context **C1** and stores it in the context management, where all its current contexts are kept (for instance also **C2** and **C3**). The participant calls the context of the conversation with the initiator "α" and stores it the same way. Context names in the figure help to emphasize that the same context instance is passed from plan to plan through the goals and between turns using the context management.

As specified by a protocol, conversations can end for agents in two ways: after sending or after receiving a message. In both cases, a goal for finishing the protocol is posted. In the first case, it happens automatically, in the second case, the goal can (in some cases it has to) be posted by the Message Attending Plan. Plans that handle these kinds of goals will contain the

internal actions to take when a participant leaves a Conversation.

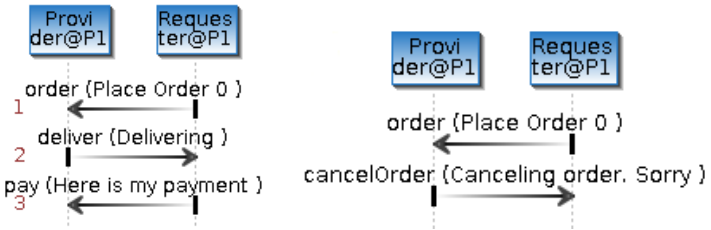
Protocol ending plans are also the place where Protocol Bindings are implemented. The conversation context is tested to be a State Description where a Protocol Binding has been established. If that is the case, propositions are translated as specified by proposition name bindings and goals for starting the next protocol with the updated conversation are posted.

6.2.4 Running the composed protocol

As an example of the results one can obtain by using the tools explained in the present work, the case of the protocols of Figure 6.9 will be used.

In the model, a composed protocol was created out of two protocols. The code generating tool takes this composed protocol and produces a Jadex agent implementation, in the form of a capability, for each of the roles involved.

Even though this is a simple example, it shows how the resulting agents behave as specified by the protocols. Figure 6.11 shows the two possible ways the protocol can be run. The diagrams show, for each of the two participants in the conversation, a vertical lane showing their execution time-line. Arrows meaning message exchanges go from side to side and are annotated with the Speech Act label of the message exchanged and in brackets, the contents of the message.



(a) Provider delivers (b) Provider cancels

Figure 6.11: Jadex run diagrams of Order-CashPayment protocol.

In (a), one can see what happens when the provider agent "Provider@P1" replies with a *deliver* message: the protocol comes to the State Description marked as **E1** in Figure 6.8 and as shown in Figure 6.9 this is where a binding with the the cashPayment protocol is defined. The Actor Payer (played by agent "'Requester@P1"') performs then pay.

In (b), one can see what happens when "'Provider@P1" replies with cancelOrder: agent "Requester@P1" stops after receiving the canceling message, since there is no further actions specified in the composed protocol for this situation. It just knows, it is left in a case where a Commitment was canceled as represented by State Description **E2** in Figure 6.8

6.3 Summary

The present chapter shows how messaging integration between agents and Web Services can be achieved and how protocols can be specified declaratively in order to produce Jadex agents capable of running as Web Services using the messaging integration tools.

The most important advantage of the proposed messaging integration is that it merges (as it can be seen in Figure 6.5 and in the example provided) information items from WS-Addressing as well as from FIPA Envelope Specification. This, at the same time, enables both technologies to connect transparently and their messages to be processed appropriately by endpoints. The usage of message addressing properties make it possible to perform complex conversations as proposed by FIPA. Agents not only take advantage of the accessibility to Web Services, as it happens with other proposals, but this implementation provides the possibility to perform complex interaction patterns using SOAP between agents and other agents or Web Services.

For achieving integration, some gaps had to be covered like the stateful nature of agents vs. stateless nature of services or the possible difference in reasoning power between participants. The first one manifests clearly in the definition of the augmented EPR, service-implementing instances are not required to be identifiable. In this implementation Web Services are presented to agents as other agents which are, or desire to be, anonymous. Agents should therefore be prepared to interact with anonymous agents and to distinguish between them by means other than its name, normally its address.

In the scope of a comprehensive agent platform modelling project called PIM4Agents a declarative model for protocols has been proposed. In the present work we have produced an Eclipse Modelling Framework meta-model and a diagram editor to produce Declarative Protocol models (León Soto 2012).

The editor shows how the protocol grows as Speech Acts are defined. After each modification by the designer, a directed graph shows what the protocol would look like, in a similar form to finite state machines. This lets the designer see how the protocol is taking shape.

Another diagram editor is also provided for producing composed protocols out of the protocols designed using this tool. By declaring Proposition Name bindings, the editor is enabled to help the user to find protocols that share compatible connection points and which can be connected together. Using Role Bindings the designer says what roles will agents of the first protocol take in the second one.

From these models, Jadex BDI agents are created automatically. The conversion takes advantage of the declarative aspect of the models, and produces a context based conversation control. BDI agents make the situation

oriented approach easier to implement than using traditional procedural techniques. At the same time, the goal oriented pattern used to implement the roles make the management of the implementation better in terms of flexibility and scalability. The result is a set of agents, capable of performing a complex conversation using Web Services.

Part III

Examples and Obtained Results

Chapter 7

Examples

This chapter presents examples that explain in detail how our tools work and highlights the features that differentiate them from other contributions from the area of multi-agent systems or business process modeling.

7.1 ContractNet Use Case

The first use case is a classical interaction pattern well known in the MAS community. It is called *Contract-Net* (Smith 1979) and represents the common situation in which an agent has a pending task and looks for another agent to perform it by broadcasting a request to a set of candidates, choosing the best one and letting it perform the task.

This protocol is part of the library of interaction protocols defined by FIPA (FIPA 2002*f*). It involves two kind of roles, the *Initiator* and the *Participant*, the first one representing one agent with a task to be solved and the second one, a set of agents that will participate in the contest for the assignment of solving the pending task.

In Figure 7.1 The first action (represented by an arrow that goes from one side to the other) is called *cfp* and stands for call for proposals. This message containing a description of the pending task is sent to a set of m *Participants* for them to analyse. Participants can answer either with a *refuse* if they do not desire to participate in the selection process or by *propos*ing a way of how to serve the requirements of the Initiator.

At the end of each arrow there is an annotation specifying the amount of agents taking part using that Speech Act. The Speech Act *cfp* is sent to m agents, n of those that answer before the deadline, $i \leq n$ send a *refuse*, the rest of them, $j = n - 1$ make a propose. The Initiator decides to send a *reject-proposal* to $k \leq j$ agents and to the rest of the agents $l = j - k$ an *accept-proposal*. All accepted agents can, at the end, send a *failure*, an *inform-done* or an *inform-result*, depending on the case, providing the Initiator with what it required.

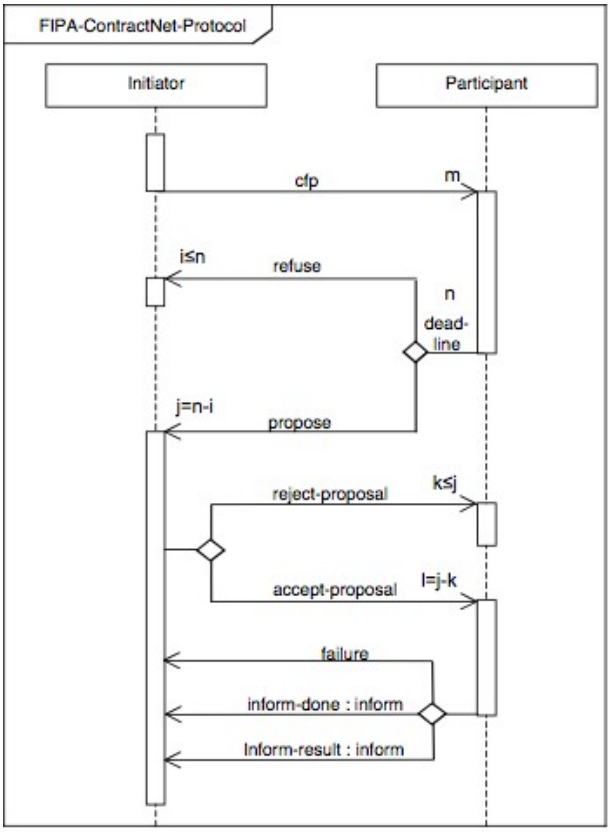


Figure 7.1: Contract Net protocol modeled by FIPA.

Contract-Net has some characteristics that make it interesting to show important aspects to be modeled:

- **Auction-like situation:** auctions are conversations in which a set of competitors make *bids* about some specific concept (in the case of Contract Net, these are called *proposals*) and one (or more) are chosen from them following a specific criteria. A protocol has to provide information about the choice that will be done like, in this case, the amount that can be chosen or when the choice is done.
- **Combination of an N:1 and a 1:1 interaction:** since the first part is where the complete set of participants interact, making bids, and in the second part, one (or a subset) of them is chosen to carry out the action. A protocol model must be capable of showing when and how this change in cardinality happens.
- **Includes several choice situations:** even though the interaction specification is not too long, at every stage of the interaction, a decision is taken, turning the protocol into one with many possible outcomes.

Using our meta-model and its modeling tools, we produced the Contract Net model shown in Figures 7.2 and 7.3. The first part is where many agents interact with the Initiator, the second is when a single one is accepted and carries out the task. In this case, we model a situation where only one Participant is chosen to exemplify in a more clear way the change in cardinality of participants ruled by the protocol.

At the top of the model is the specification of *Proposition Names* to be used in the protocol:

- **cfpAttended:** has the *cfp* been attended in any way?
- **accepted:** has the proposal been accepted
- **prop-Successful:** has the proposal been carried out successfully
- **Wait(cfpAttended):** wait for all *cfp* to be attended
- **prop-Evaluated:** has the proposal been evaluated (regardless of the result of the evaluation)?
- **accept-Attended:** has the acceptance been attended in any way?
- **Timeout(reject-proposal):** Timeout by the end of which a *reject-proposal* is carried out passively on behalf of the Initiator.
- **Timeout(failure):** Timeout by the end of which a *failure* is carried out on behalf of the Participant.
- **Commitment(prop-Successful⇐accepted,Participant → Initiator, Timeout(failure)):** a Commitment to bring about *prop-successful* (before the Timeout) if the proposal is *accepted* from the Participant to the Initiator. At the end, if the time runs out, a *failure* is carried out as specified by the Timeout.
- **Commitment(cfpAttended, Participant → Initiator):** a Commitment to bring about *cfpAttended* from the Participant to the Initiator with no Timeout specified.
- **Commitment(prop-Evaluated, Initiator → Participant, Timeout(reject-proposal)):** a Commitment to bring about *prop-Evaluated* from the Participant to the Initiator before the time runs out, otherwise *reject-proposal* will be performed on behalf of the Initiator.
- **Commitment(prop-Successful, Participant → Initiator, Timeout (failure)):** a Commitment to bring about *prop-Successful* from the Participant to the Initiator before the Timeout, otherwise a *failure* is performed on behalf of the Participant.

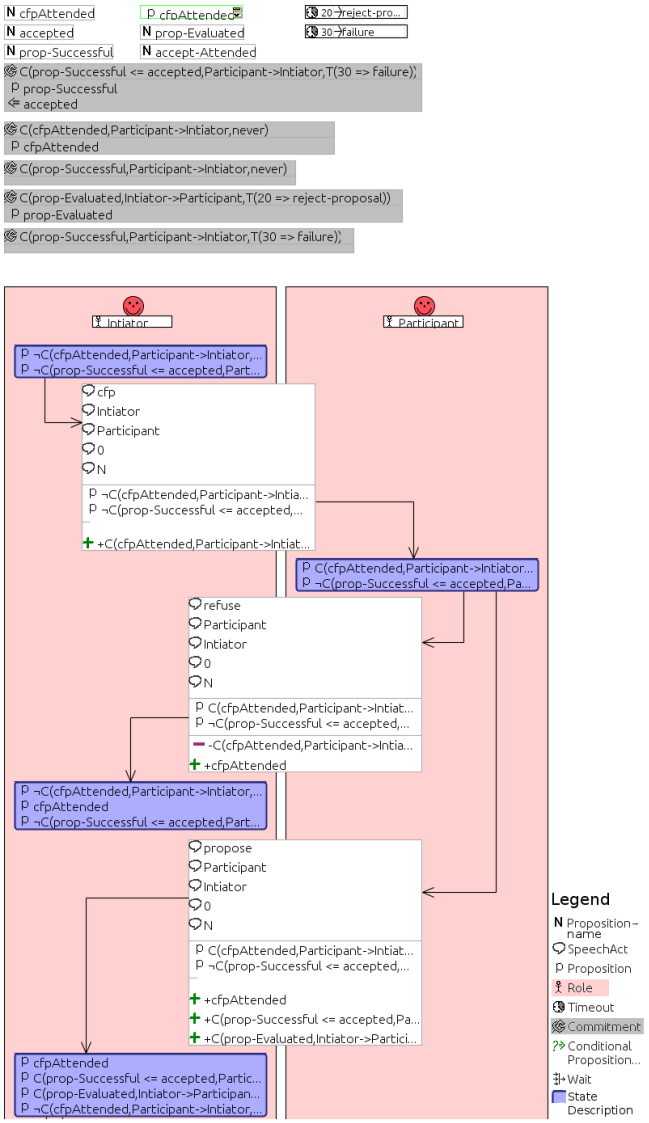


Figure 7.2: Contract Net protocol modeled as a Declarative Protocol. First Part

At the beginning, an Initiator starting from a situation where no Commitment has been created, sends a *cfp* to N Participants. Sending this action creates a Commitment from the Participant to the Initiator to attend the call. This Commitment has been left without a Timeout for simplicity. If the Participant responds with a *refuse*, it frees itself of the Commitment created by the *cfp* by bringing about *cfpAttended*. If it makes a *propose*, the cfp is attended (freeing the Participant of the corresponding Commitment based on the specification defined in section 5.4.2) and creates new

Commitments: one where the Initiator makes a compromise to evaluate the proposal and provide an answer and one for the Participant to make the proposal successful, if the proposal is accepted. At the end of the first part of the Contract Net protocol diagram, the State Description (blue rectangle) on the bottom left shows the state of the conversation. This same state appears in the next figure on the top left, to help the reader follow the link between the two parts.

The second part, shown in Figure 7.3, shows that the Initiator can opt to send a *reject-proposal* or an *accept-proposal* message. The first one making it clear that the proposal is not accepted (-*accepted*) and also brings about *prop-Evaluated* canceling the related Commitment. The second one, brings about *accept* and *prop-Evaluated*, furthermore, it clarifies that the accept has not been attended (-*accept-Attended*).

In this situation, the condition of the conditional Commitment brought about by the action *propose* is made true, which automatically makes the Commitment of the Participant to the Initiator to bring about *prop-Successful* valid. Additionally, the associated Timeout is started, stating that after 30 time units, a *failure* will be performed on behalf of the Participant. The action *accept-Proposal* has special cardinality constraints, different than the default 0-N: it has a minimum of 1 and a maximum of 1. This means that the Initiator can only use this action once, letting it choose only one and at least one of the proposals.

The actions *accept-* and *reject-proposal* have one particular precondition different than the trivial ones: *Wait(cfpAttended)*. It restrains the sender from going on, until all conversations it is managing bring about the proposition *cfpAttended*, which is achieved by any reply-actions to *cfp*. This construct makes the Initiator wait for all Participants to react in some way to the *cfp* and allows it to proceed to *accept* and *reject* after all participant have replied.

A *wait* condition like this one is a key feature introduced by our metamodel, which was not present in previous proposals. It represents an inflection point in the protocol, where all conversations that in principle run parallel and independently of each other are held and *synchronized* by the Initiator. It is the statement in the protocol that guarantees that a decision will not be announced before participants provide their proposal, if they do it within the designated time. A protocol can only specify aspects related to the conversation that are visible from the outside. Although it prohibits the Initiator sending an *accept* before all *propose* and *refuse* have been received, it cannot force the Initiator agent to internally decide ahead of time and before all replies have been received. Such a ruling is considered to be out reach of an interaction protocol model.

Once the Participant has been accepted and a *prop-Successful* is being expected, it can perform an *inform-result*, honoring the Commitment (making it false), since this action brings about *accept-attended* and more im-

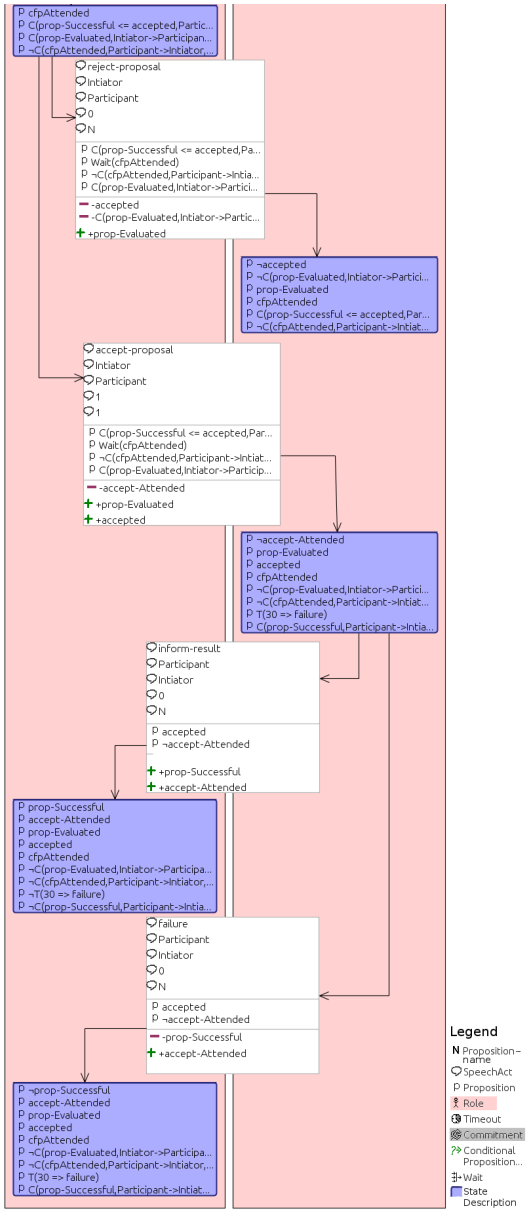


Figure 7.3: Contract Net protocol modeled as a Declarative Protocol. Second part.

portantly *prop-Successful*. Performing failure would make *accept-Attended* true and *prop-Successful* false. The second one, dishonoring the Commitment accepted by the Participant, producing a State Description in which the Commitment to bring about *prop-Successful* is valid, but its objective, *prop-Successful* is invalid. A clear situation, in which the agent has broken its promise. This is an ending State Description and therefore the protocol

does not define what is to be done with agents that end up in such a situation, but provides at least a concrete connection point where further actions or protocols can be connected to specify how to proceed in those situations.

Other ending State Descriptions are those after the actions: *inform-result*, *reject-proposal* and *refuse*. These State Descriptions along the starting State Description located before the action *cfp* are the connection points of this protocol to other protocols in a composition scenario.

The next example shows how protocol composition works, highlighting some of its advantages.

7.2 Industrial Use Case: Saarstahl

The following is an industrial use case that defines a real-life business process to show how our tools perform in a more concrete and realistic scenario. This use case is part of a European research project called SHAPE (Benguria, Berre, Elvesæter, Hahn, Jacobi, Landre, Sadovykh & Stollberg 2009), where agent oriented software development was used, among other approaches, to model business processes.

This use case provides a real-life scenario, proposed by Saarstahl AG (Saarstahl AG n.d.), a Steel producer in the region of Saarland, Germany and partner in SHAPE. This use case has been created specially to evaluate service-oriented modelling and implementation tools and is therefore very suitable to test our proposal.

In the following sections, the company, the project, and the use case will be presented. After that, the way this use case is modelled and implemented, using the tools provided, as part of this thesis, will be shown.

7.2.1 Research and industrial partner: Saarstahl AG

Saarstahl AG(Saarstahl AG n.d.) is a steel producer based in the region of Saarland and more precisely in the city of Völklingen, with other production sites in the nearby cities of Neunkirchen, Burbach and Nauweiler. It specializes in high quality steel *long products* like wire rod and steel bars.

The complete production process inside Saarstahl involves a big supply chain which is depicted in Figure 7.4. Its Steelwork allows Saarstahl to produce steel alloys out of pig-iron provided by an external supplier. Iron is converted into various kinds of steel alloys and casted into *cast blooms* of different formats. These can be processed in rolling mills to produce other products like wires and steel bars. The products can also be subjected to other supplementary treatments like arrangements, pickling, annealing and cutting which modify them in form, precision, or surface properties.

The variety of products are supplied to customers in a wide range of branches, which go from automotive suppliers, through energy producers to aerospace industry.

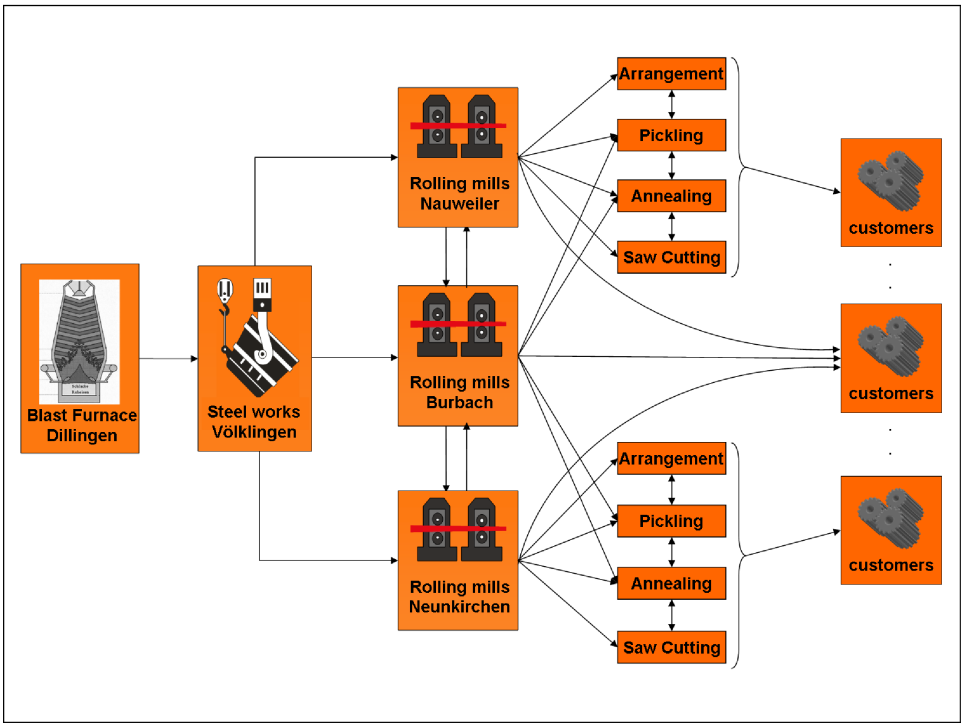


Figure 7.4: Saarstahl Supply Chain

Saarstahl's production chain is characterized as different from the mainstream supply chains, because it has a very reduced amount of input sources and a wide variety of results. In principle, there is only one source of material for its products: pig iron (ignoring the various other products necessary to produce steel alloys). As Figure 7.4 shows, from the same product a variety of products can be produced by combining different parts of its production sections. What is not possible to represent in this figure, is the path products have throughout their production. Different orders are put together to form batches for a specific step in their production. Afterwards, they are separated to take each order to its subsequent production step. Here they are put together with other orders that share that same requirement. The production paths of the different items form a labyrinth that is not straightforward to manage. For instance, pig iron is converted to steel alloys in charges that weight around 170 tons. Many different items will be produced from this charge of steel. Once it is casted, parts of this charge will move on through different paths for further production. Some of these paths will, for instance, join other steel products with different characteristics that need the same treatment. These are then batched together in this stage of production and separated again afterwards, for each to continue on in its own path.

Looking at the production chain from a global point of view, Saarstahl, being a raw material supplier, is at a position at the beginning of the production chain and hence, is severely affected by the bullwhip effect (Forrester 1969), which happens when participants in a production chain have extra stock to prevent problems with demand fluctuations. As a result, these fluctuations increase as we move back in the supply chain, causing big demand fluctuations for suppliers at the beginning of the chain. Saarstahl sees, in new technologies, the chance to improve their supply chain management systems, in order to enhance their capabilities of coping with the bullwhip effect by having more flexibility and reducing their need to add precautionary stock.

These conditions result in a production scenario, not easy to model using conventional industrial engineering techniques and in some cases, new solutions are required. In some of them, using multi-agents technology has provided advantages.

Saarstahl enjoys of a high level of competence in the field of steel alloys, because of its continuous commitment to the highest technology and knowledge in the field. A symbol of its dedication to remain in the leading sector of its branch is, other research projects carried out related to the research in this thesis:

- AgentSteel (Jacobi, León Soto, Madrigal Mora & Fischer 2005): multi-agent technologies in the steel production, preliminary research on usage of MAS technology in the management of supply chains.
- DISPO (Jacobi, León Soto, Madrigal Mora & Fischer 2007): A multi-agent system for planing and monitoring production in a steelwork.

Due to their interest in improving its production management, Saarstahl participated in SHAPE (Benguria et al. 2009), a European research project which will be presented next:

7.2.2 SHAPE: Semantically-enabled Heterogeneous Service Architecture and Platforms Engineering

Shape (Benguria et al. 2009) is an European project (EC FP7) targeted at providing an integrated development environment (IDE) for SOAs using MDA techniques. The idea is to provide innovative service engineering methods characterized by being flexible and customizable like adaptive systems, semantic technologies and agent technologies.

The motivation for this is that modern organizations in the industrial sector are forced to improve their production process, in terms of efficiency, in order to maintain their position in the market. Keeping a superior product quality alone is, by far, not enough.

At the level of computer systems supporting these processes two clear trends have been established: SOA and MDA. The first one helping integration between organizations and heterogeneous systems. The second one is related to the way process are designed, with different aspects treated in different layers, from abstract business process models to concrete implementation models. The two-fold structure of this thesis matches these two trends.

Its modeling aspect is based in SoaML (*Service oriented architecture Modeling Language* 2008), a standard meta-model for SOAs driven by OMG.

7.2.3 Description of industrial Use Case: Saarstahl

The tools and techniques provided in the scope of SHAPE have been tested and evaluated in Saarstahl AG with the goal of improving the integration of legacy systems, isolated solutions and new agent-based technologies and other types of solutions that compose their process support systems.

Saarstahl's internal production chain is very large. It exceeds the extension planed for project SHAPE and therefore a sub-part of it was selected to be used as a use case: the interaction between steelwork and rolling mills.

Most of what the steelwork produces is to be processed at rolling mills. The stocks placed between the two, steelwork and mills, are called *semi-finished* product. Coordination between these two phases in production represents a vast part of the coordination work done in Saarstahl. The objective is to coordinate the semi-finished product in such a way that it is always produced close to when it will be needed in the mills, but without delaying its process beyond the contracted delivery date.

Saarstahl System Landscape

The landscape of systems involved in the use case is shown in Figure 7.5 (Raber 2009), where the three parts of the production chain are represented by the following computer systems:

- **Steelworks Planning System**: it is composed of production data management systems and planning systems which aim production planning at different levels of abstraction. From preliminary distribution and sequencing of orders in charges to detailed low level scheduling of the production in the steelwork (Jacobi et al. 2007).

- **Semi-finished Product Management System**: it is in charge of managing the semi-finished product resulting from Steelwork, waiting to be processed in the mills and managing the warehousing. Every time an item is requested to be treated in the mills, matching semi-finished products are looked for. In case there is no material in stock, the request is forwarded to the steelwork.

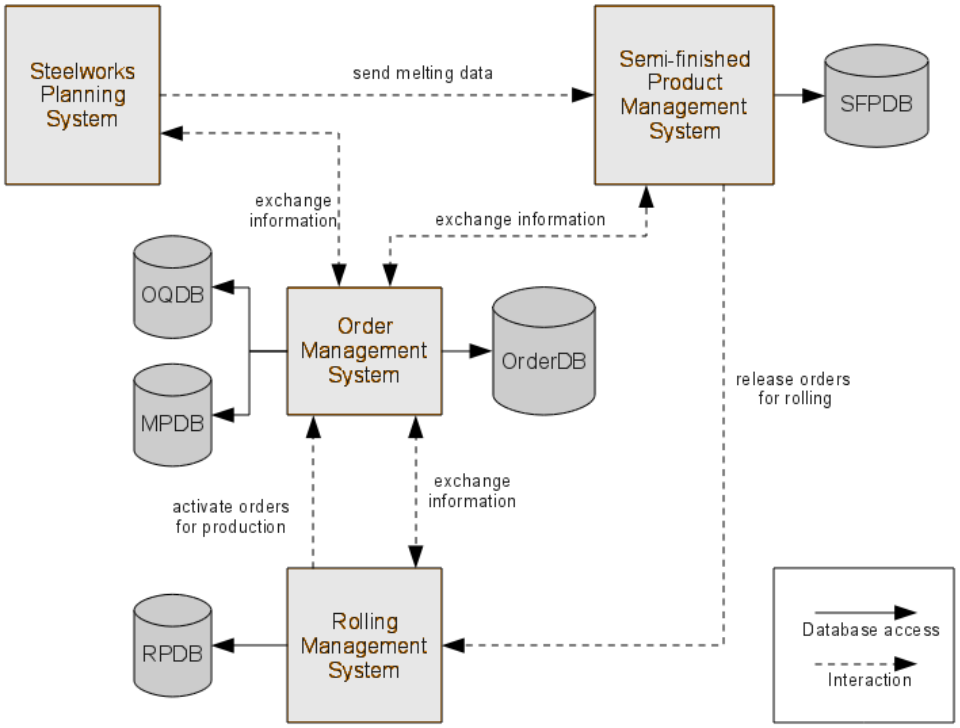


Figure 7.5: Saarstahl System Landscape (Rabber 2009)

- **Rolling Management System**: it schedules orders in rolling campaigns: products that share the same type of process are scheduled together to form a campaign. Campaigns are performed in cycles that minimize production-settings changes, which, in the case of rolling mills, involve relevant costs.

The name of the databases they access is also included in the figure. There is a forth system that takes part in the landscape of the use case: the **Order Management System**: it is mainly required to manage deadlines and product specifications that have been agreed upon with clients.

This scenario involves very heterogeneous systems, not only concerning technology, but also age. Some java based cutting edge MAS-based planning systems like MASDispo coexist with legacy systems, some of them being very old, for instance: COBOL systems form part of the order management system. Communication is also done using different, non-standardized interfaces based on FTP, XML file exchanges and even bare TCP packages.

The objective of the use case in the project SHAPE is to prove the ability of model driven SOA to integrate heterogeneous systems. Saarstahl aims at implementing a modern integration of its systems that matches its requirements.

Use case story line

The process to be modelled starts with order requests being posted by clients to the sales department. These orders are normally composed of different goods and have to be broken down into in its production components. The same components of different orders are collected in sets big enough to completely exploit the production capacity of aggregates.

Collected sets of goods of the same kind that are to be produced together are called *charges* in the steelwork and *campaigns* in the rolling mills. Charges are in fact 170 tons of melted steel contained in a ladle and are the optimal working unit for the aggregates in the steelwork. In the steelwork, the pig iron in the ladle is turned into the desired steel alloy and casted to solid bars. Campaigns instead, are sets of solid goods that are treated in row, one after the other, in order to take advantage of the settings of the treating aggregate. Changing settings normally involves added costs, since it means temperature adjustments. Minimizing these adjustments (to only one or two every day) improves efficiency significantly. Some campaigns can reach a span of a week more or less.

The sales department forwards the components to the production planning department to find out if the order can be delivered by the deadline agreed with the client. This is done by checking coarsely if resources are still available, which means, looking for either; available inventory of semi-finished goods that match the requirements, or available space in the corresponding working sets in the near future.

Once it has been determined that the request can be served and the order is placed, concrete order planning is carried out by assigning the found inventory to the order, or assigning concrete allocations for the goods.

First, melting jobs are scheduled in the Steelwork for all the material needed to complete the order that could not be supplied with existing stock. Once the material is produced, it has to be tested to meet the requirements. If that is not the case, the charge is left in the semi-finished warehouses as available stock for future orders and production of new material is scheduled again.

Once the amount of material is completely available to start milling, the order is schedule in the next possible campaign, which, for the terms of this use case, concludes the story line.

7.2.4 Use case modelled using Declarative Protocols

The previous story line has been used to develop a protocol model for the Saarstahl Use case. The approach has been divided in two parts:

- A pre-order phase (shown in Figure 7.8), in which a client asks whether some kind of order is possible.

- If the first part comes to an agreement, an order phase is started by the client (shown in Figure 7.8), in which it places the concrete order.

The protocol was implemented using two protocol models used as construction bricks:

- **Order:** the order protocol is a non standard protocol created to model the interactions between a client, sales, and planning departments in Saarstahl.
- **QueryIf:** is a FIPA standard protocol used to request some information or task? In our case it has been used to model the queries Planning does to the Semi-finished, Steelwork and Mills departments. In the second part it is also used to instruct each department to produce their part of the order as agreed in the first part of the protocol

This model has been produced making use of the composition capabilities of our meta-model. It is illustrated in the following set of figures.

Composition model of the first part

As shown in Figure 7.8, on the top left is a protocol called Order_preorder. The meaning of the name is; it is an Order protocol used in the context of a pre-order. In this layer, we are only interested in the connection possibilities of the protocol. In the figure, the various possibilities are additionally annotated to understand their meaning.

The Order_* protocol is a protocol specifically created for this use case and it involves 3 roles: client, sales and planning. When it is started, a client asks a sales agent if it is possible to produce a certain order. Sales requests confirmation from the planning agent. At this point, represented by the state with the label "Planning Consulted", an ending State Description is defined in order to be able to connect here: protocols that enable planning to resolve the information request. It does so by performing a sequence of queries to the different departments of the factory. Depending on the answers it collects, it confirms or denies the possibility of producing the order.

On the left, there are three entry points or starting State Descriptions. At the very top, the actual entry point of the complete model. On the right there are several exit points of the protocol. At the top right: a group of four State Descriptions that represent failure situations caused by the sales or planning agents. Bellow that, there are three other exit points. The first of the three is the state in which the planning agent has been consulted and it moves on to the sequence of queries it does to the components of the factory.

Once the queries are finished, the conversation is lead to the two other entry points on the left of Order_preorder. One starting State Description

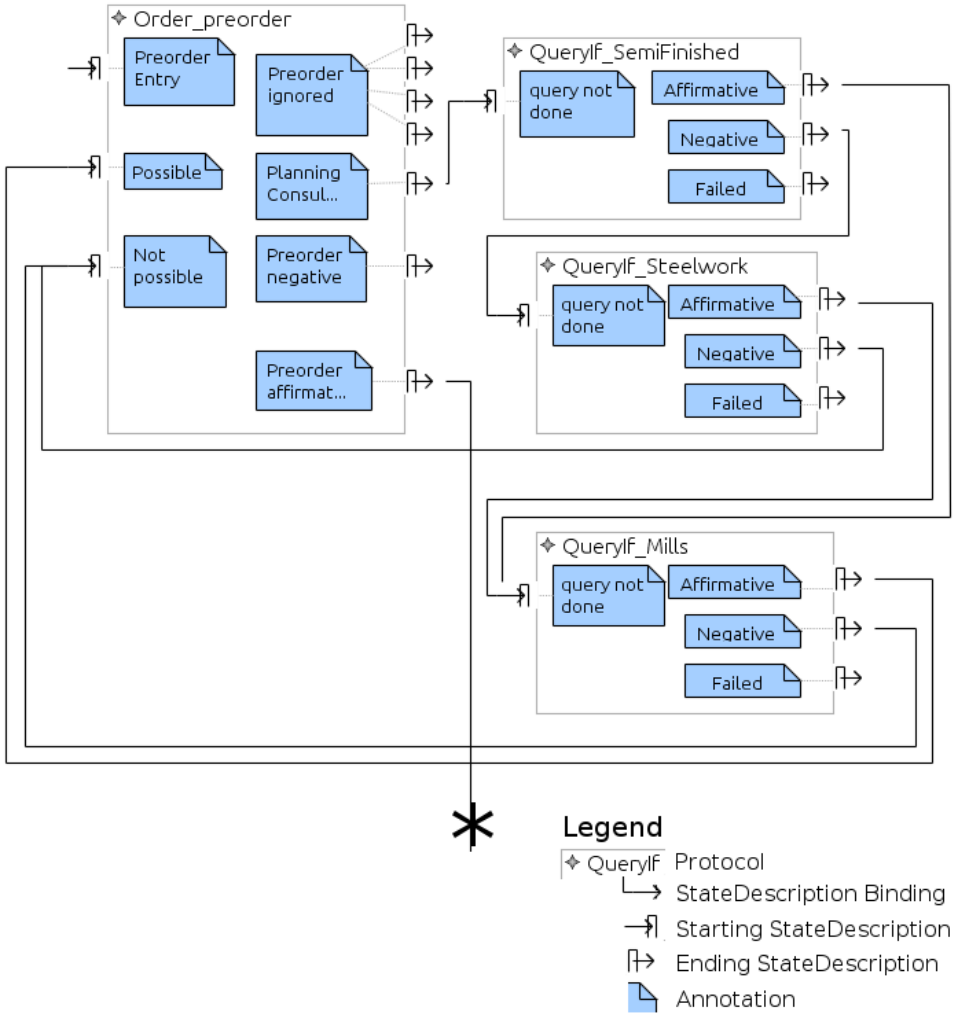


Figure 7.6: Saarstahl Use Case project diagram. First part.

is for when the production of the order has been reported as possible and the other one for when it is not possible. Analogously, there are two other ending State Descriptions, one to represent the situation in which sales has confirmed the preorder as possible and one for when it is not possible.

In case the order is positively confirmed, the protocol composition leads the conversation to the next part, represented with a big asterisk as a connection point.

On the right hand side, again, are the sequence of queries that planning does to the three departments in the factory. Once planning is consulted in the Order_preorder protocol the conversation is lead to the QueryIf_SemiFinished protocol. This is one of three identical QueryIf protocols that are used to interact with each factory department. Depending on the answer received

from SemiFinished, the conversation is lead to query the Steelwork, in case there is not enough semi-finished product, or directly to Mills. After querying the Steelwork and getting a positive answer, the conversation is lead to the mills query. Otherwise, it is lead to the "not possible" entry point in the Order_preorder protocol. After querying the Mills, the conversation is taken to the "possible" and "not possible" entry points of the Order_preorder protocol, depending on the answer of the Mills being affirmative or not.

This first part already shows the advantages of a protocol meta-model that provides modularization possibilities. The queries section has been implemented using the same protocol definition three times, once for each factory department. Also, being able to modify the connections between the protocols makes the model very flexible and scalable.

As it will be shown next, the second part takes even more advantage of this approach, since it reuses the same models to represent its process.

Composition model of the second part

The second part, as can seen in Figure 7.7, is a repetition of the first part. In this case, only the contexts are changed in order to differentiate them from the first part, but the protocol "types" are the same: one Order protocol and three QueryIf protocols. In this case a semantic difference has to be clarified concerning the QueryIf protocols: they do not represent a typical query situation as those that appeared in the first part, but actually represent assignation of tasks to each of the factory departments. Since the possibility of producing the order has been confirmed in the first part, there should not be any impediment to produce it. Even so, the protocol has been modeled using the QueryIfs, to provide a mechanisms to communicate if something goes wrong in the production of the order.

The connections and structure of the second part is identical to the first part, it has been labeled differently to those in the first part to tell them apart: in this case the order protocol is called "Order_order" since it actually orders a product, the QueryIfs are suffixed with a "2", to mark them as the ones used in the second part.

Following the connection represented by the asterisk on top of the figure, one can see that the second part starts also through the same State Description. Here, the conversation takes the same course, as it does in the first part. The ending State Description in the Order_order protocol labeled "Order positive" is the actual ending State Description of the composed protocol, where the whole conversation came to a positive end: the product has been delivered.

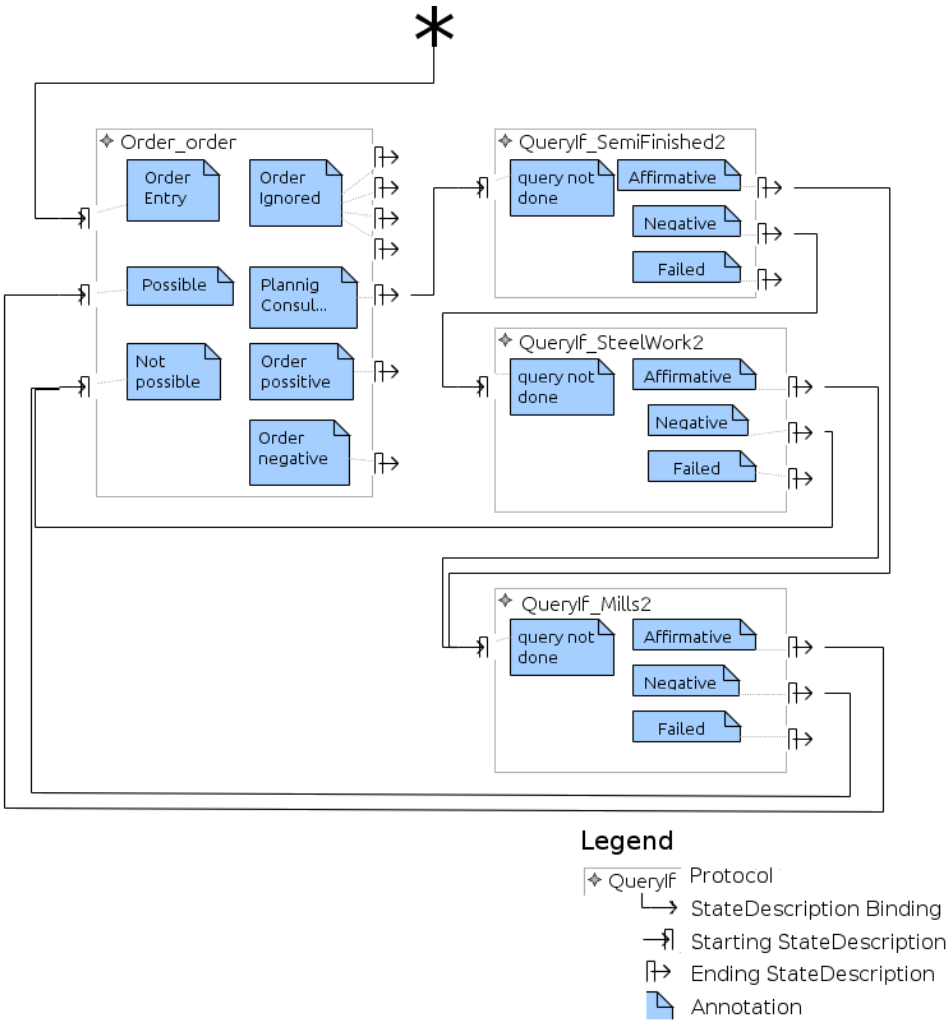


Figure 7.7: Saarstahl use case project diagram. Second part.

Protocol view of the Order_order protocol

As already mentioned, the composed protocol is the result of recombining two types of protocols, the Order protocol and the QueryIf protocol. In this section we will have a look at the details of the Order protocol by looking at its protocol diagram. Due to reasons of space, the diagram has been divided again in two figures: Figure 7.8 and 7.9.

In Figure 7.8 one can see at the top the concepts used to define the protocol: a set of simple Proposition Names, Timeouts and Commitments that will be explained next (the suffix _order will be omitted):

- **orderPlaced:** That the order has been placed.
- **orderAttended** That the order has been attended, either by accept-

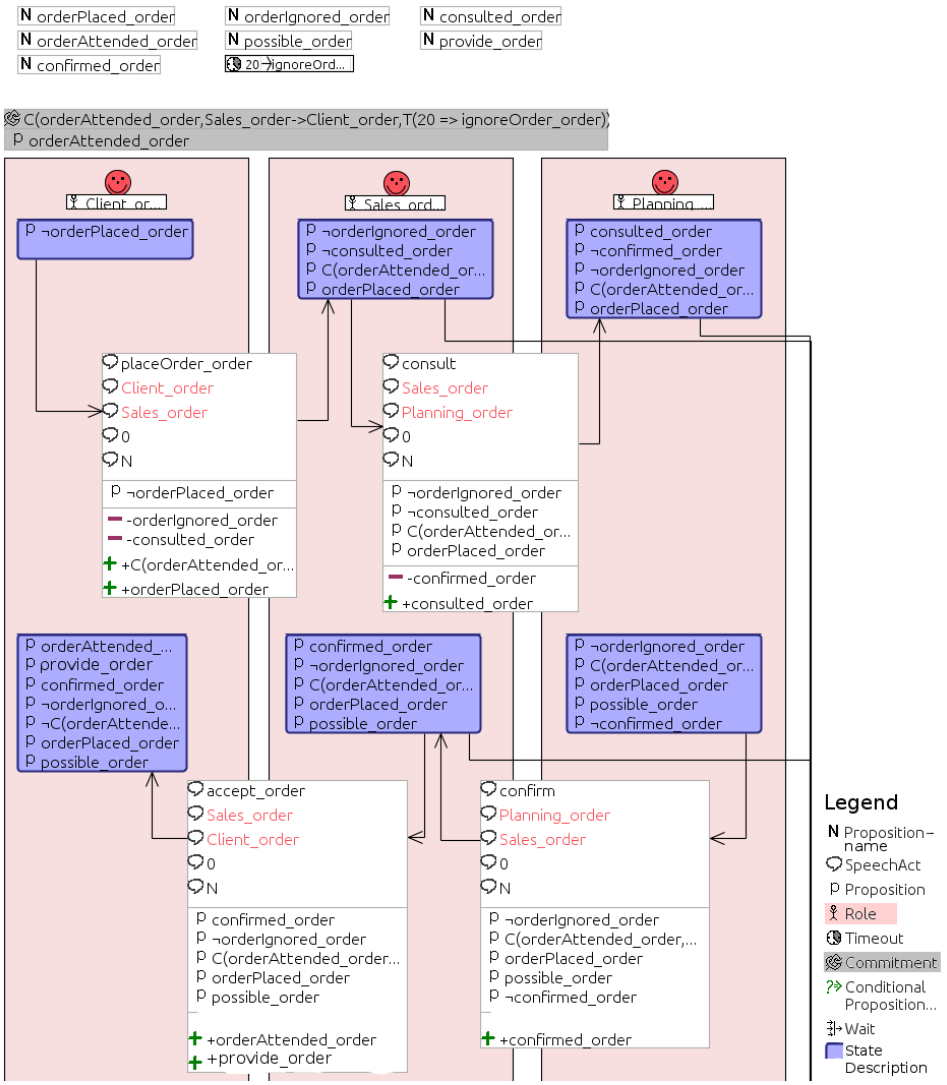


Figure 7.8: Order protocol. First part.

ing or refusing it.

- **confirmed:** The order has been confirmed by the factory.
- **orderIgnored:** The order has been ignored.
- **possible:** Production was possible for the order.
- **consulted:** Planning has been told about producing the order.
- **provide:** The order has been provided.

- **Timeout(Ignored)** Timeout for Sales to answer the order request of the client.
- **Commitment(orderAttended, Sales→Client):** the Commitment to attend the order request of the client.

The diagram has been enhanced with three vertical rectangles in the background (light red) representing a common concept in modeling interactions, called "swimming lanes". They help to better understand which role has the turn in a specific State Description, placing the State Description in the swimming lane of the corresponding Role.

The protocol starts on the top left, with a situation in which the order has not been placed ($\neg orderPlaced$). In this case, a Client can place an order by making use of the action *placeOrder*, which makes $+orderPlaced$ true, also enables the Commitment to attend the order ($orderAttended$) and establishes that it has not been ignored ($\neg orderIgnored'$) or *consulted*. This leads to a State Description in which Sales can react and forward the consult to Planning. In the case of "preorder" this is indeed a consultation, but in case of the "order" (the case in the Figure), it is to be understood more like the command to start producing the order.

This leads to a State Description that has a particularity: it was designated as an *extra ending State Description*. Even though there is an action that starts from this State Description (something ending State Descriptions normally do not have), this one has been specially marked as such, in order to be capable of connecting other protocols at this point in the conversation. This State Description is the ending State Description annotated as "Planning Consulted" in the composition view.

In this figure, there is on the right hand side another starting State Description (Annotated as "Possible" in the composition view in Figure 7.6). It describes a situation in which the order has still not been ignored ($\neg orderIgnored$), the Commitment to attend the order is still valid, the order has been placed and most importantly: the order has been designated as *possible* (in the case of _preorder it means indeed that it is possible to produce it, in the case of _order it means it was successfully produced), but it has not been confirmed ($\neg confirmed$). The action *confirm* lets Planning inform Sales about this fact, making $confirmed$ true. At this State Description, Sales can *accept* the order (or in the particular case of _order to inform the successful completion of the order), making *provide* true.

Exactly the opposite situation is described in the last part of the protocol (Figure 7.9): another starting State Description (annotated as "Not possible" in the composition view, Figure 7.6) represents again a situation like the one previously described, with the only difference that it is not possible to attend the order ($\neg possible$). In this case Sales is only capable of replying with a *refuse* message, making only $orderAttended$ true, but not *provide*.

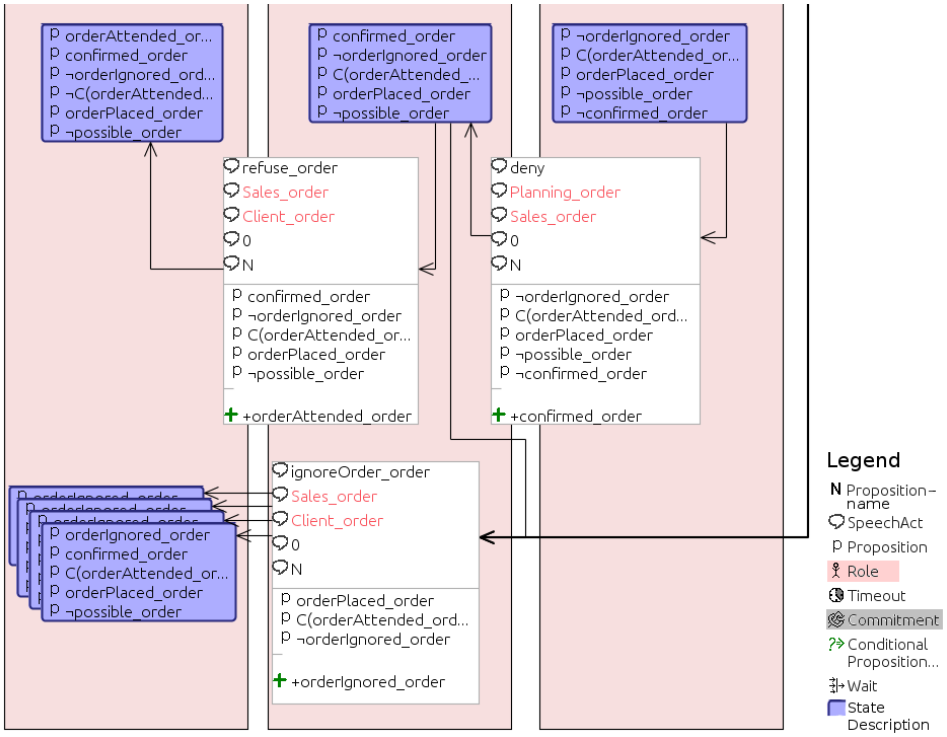


Figure 7.9: Order protocol. Second part.

Bringing about *orderAttended* makes the editor automatically disable the associated Commitment. The result of this can be seen in the ending State Description on the top left of the figure.

On the lower part of the figure, one can see the action associated with the Timeout related to attending the order. If Sales takes longer than what the Timeout specifies, then the order is regarded as ignored by the client. This is modeled using the action *ignoreOrder*, which is enabled in almost any situation where Sales is active and correspondingly produces many different results, depending on the state the conversation was in. For the purpose of the protocol model, the fact that the order has been ignored is the only thing relevant if this happens. That is why all these ending State Descriptions are collected in a Stack. These ending State Descriptions are also collectively shown in the composition Diagram and annotated with the text "Order ignored".

Protocol view of the QueryIf_Steelwork protocol

The rest of the composed protocol is done using 6 different kinds of "QueryIf" protocols: one for each factory department: Semi-finished, Steelwork and Mills in the first part (pre-order) and the second part (order).

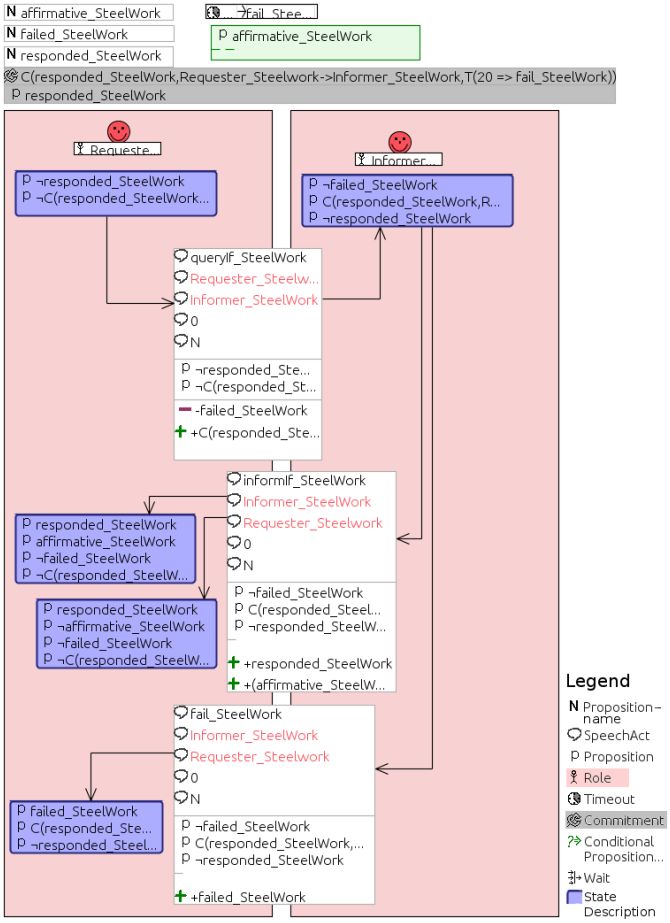


Figure 7.10: QueryIf protocol

All these protocols share the same internal structure. Figure 7.10 shows the protocol diagram of the QueryIf done to the Steelwork in the first part of the composed protocol.

As usual, on the top are the declaration of Proposition Names used in the protocol:

- **affirmative:** the answer to the query is affirmative.
- **failed:** the Informer failed to answer.
- **responded:** the query has been responded to, regardless if affirmative or negative.
- **Timeout(fail):** Timeout to respond before a *fail* is assumed.
- **Conditional(affirmative):** a conditional proposition name, in this case without consequences. It means that, based on this proposition,

two conditions are to be calculated: one for $affirmative$ and one for $\neg affirmative$.

- **Commitment(responded, Informer →Requester):** the Commitment from the Informer to the Requester to make *responded* true, in other words, to react within the expected time, before a "fail" is assumed by the Requester.

The protocol starts in a State Description where nothing has been responded and there is no expectation of something being responded. By making *queryIf*, the Requester enables the Commitment to attend the query and at the same time makes it clear that it still has not failed ($\neg failed$). After that the Informer can reply with an *informIf*, which has two possible outcomes: one with and one without negation. The action *fail* can be actively sent by the user or is assumed, if the Timeout runs out of time.

The starting State Description on the top left is the one annotated with "query not done" in the composition view. The two results of *informIf* are annotated with "Affirmative" and "Negative" and the last ending State Description at the bottom left is represented by the one annotated with "Failed" in the composition view.

Running the Saarstahl Use Case

The model represented by the diagrams presented here can be transformed to executable code for Jadex. The resulting code has to be completed with the details about the concrete scenario, like the deployment of actual agents that take the roles (made to capabilities) and the actual content of the messages, among other details.

Running the resulting code produces the diagram in Figure 7.11. This is the particular case in which Semi-finished does not have enough stock to supply the order and the Steelwork is requested to provide the necessary steel. Once everything has been confirmed to be possible, the client proceeds to place the actual order, which repeats the whole interaction again, this time to produce, in fact, the order.

The diagram is organized again in a "swimming lane" pattern, where each agent gets a column: the order of the agents cannot be customized and therefore each name will be explained next. Note that each name ends with a 1 to represent the idea that it is a concrete and particular *instance* of an agent. The name of the platform is "SaarstahlUseCase"

- Client1@SaarstahlUseCase
- Mills1@SaarstahlUseCase
- Planning1@SaarstahlUseCase

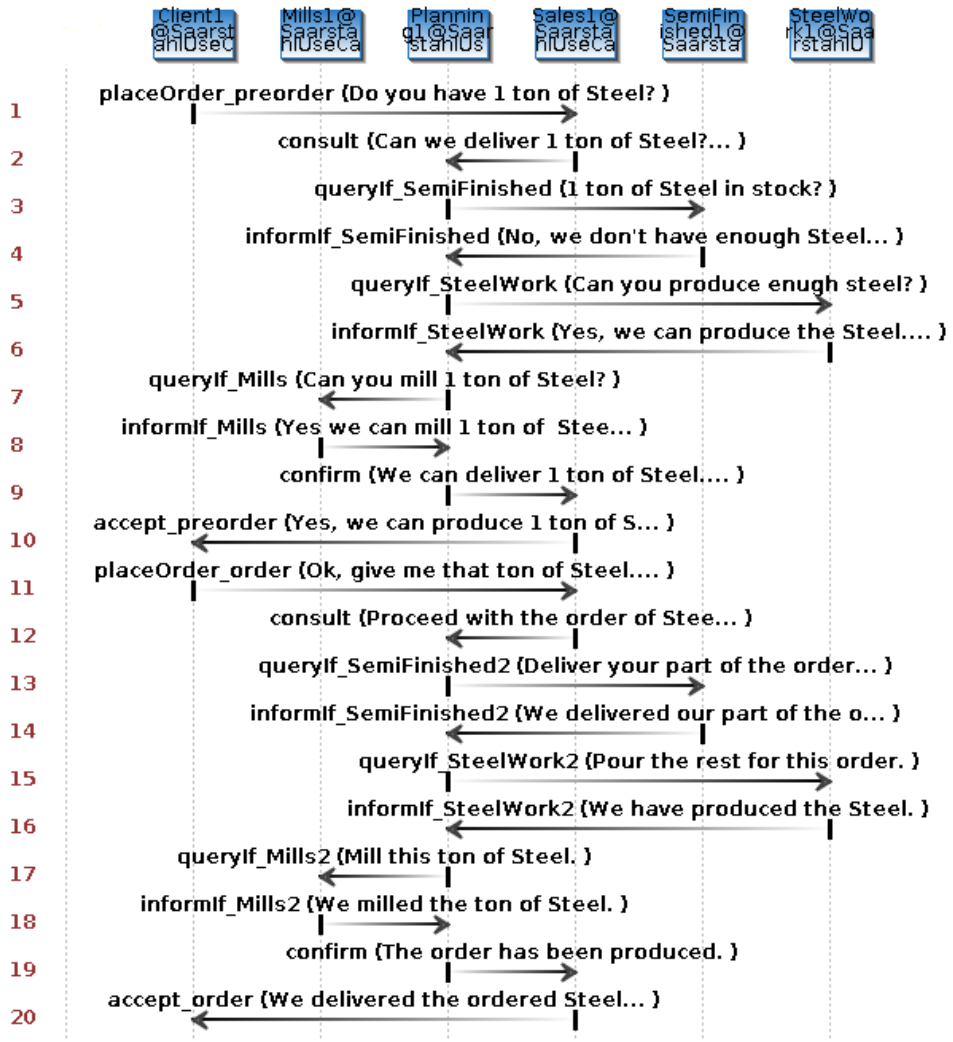


Figure 7.11: Interaction Diagram of the Saarstahl use case.

- Sales1@SaarstahlUseCase
- SemiFinished1@SaarstahlUseCase
- Seelwork1@SaarstahlUseCase

In each line a message interchange between two agents is represented by an arrow, which is annotated with the name of the Speech Act and its contents.

A detailed explanation of what happens in each line follows:

1. **placeOrder_preorder:** Client1 asks Sales1: "Do you have 1 ton of Steel?"

2. **consult:** Sales1 consults Planning1: “Can we deliver 1 ton of Steel?”

3. **QueryIf_SemiFinished:** Planning1 queries SemiFinished1:“(Do you have) 1 ton of Steel in Stock?”. SemiFinished1 is now expected to check its stock and answer, whether it can provide the steel for the order or not.

4. **InformIf_SemiFinished:** SemiFinished1 informs Planning1: “No, we don’t have enough Steel”.

5. **QueryIf_Steelwork** therefore Planning1 queries Steelwork1: “Can you produce enough Steel?” so that there can be enough Steel to serve the order.

6. **InformIf_Steelwork:** Steelwork1 informs Planning1: “Yes we can produce the steel”, letting Planning1 know that it now just needs to confirm with mills.

7. **QueryIf_Mills:** Planning1 queries Mills1: “Can you mill 1 ton of Steel?”

8. **InformIf_Mills:** Mills1 informs Planning1: “Yes, we can mill 1 ton of Steel.”

9. **confirm:** knowing that it can be produced as desired, Planning1 proceeds to confirm Sales1: “We can deliver 1 ton of Steel”.

10. **accept_preorder:** this lets Sales1 accept the order to Client1: “Yes, we can produce 1 ton of Steel.”.

11. **placeOrder_order:** Client1 places the order to Sales1: “Ok, give me that ton of Steel.”

12. **consult:** Sales1 consults Planning1: “Proceed with the order of Steel.”

13. **QueryIf_SemiFinished2:** Planning1 queries SemiFinished1:“Deliver your part of the order”.

14. **InformIf_SemiFinished2:** SemiFinished1 informs Planning1: “We delivered our part of the order.”.

15. **QueryIf_Steelwork2** Planning1 queries Steelwork1: “Pour the rest of the order.”

16. **InformIf_Steelwork2:** Steelwork1 informs Planning1: “We have produced the Steel.”.

17. **QueryIf_Mills2:** Planning1 queries Mills1: “Mill this ton of Steel.”

18. **InformIf_Mills2:** Mills1 informs Planning1: "The order has been produced."

19. **confirm:** knowing that it is complete, Planning1 proceeds to confirm Sales1: "The order has been produced".

20. **accept_order:** this lets Sales1 deliver the order to Client1: "We delivered the ordered Steel.").

The generated Jadex agents are standard BDI Jadex implementations and hence can be run over the Web Services enabled platform produced in section 6.1. Providing an implementation of the protocol model using Web Services. The transparency of the Web Services tool lets us use the produced code with very little changes: basically, agents have to specify to use SOAP as message encoding, to use the Web Services communication and that the deployment details were adjusted, to match the different scenario: addressing has to be updated because the agents are in different platforms.

7.3 Summary

Due to the amount of details that show up in a normal situation using the models and tools part of our work, it was necessary to dedicate a Chapter specially for examples where the complete usage scenario could be presented. We have provided two scenarios: a classical use case for agent interaction protocols: the Contract Net and an industrial use case designed to test modelling and SOA realization tools for business processes.

The Contract Net showed among other details, the necessity of having the possibility to manage parallel conversations that have something in common. Our model provides clear and concrete features to specify, where a conversation can run freely in parallel to others, and where a participant is expected to wait for others to respond in order to continue the conversations.

The industrial use case is provided by SHAPE, a European Research Project this thesis is part of. It was specified by Saarstahl, our industrial partner, to use SOA and MDA to integrate their very heterogeneous system landscape. Our example shows that it is capable of modelling and realizing the use case with more ease and reusing vasts amounts of concepts. The examples prove our meta-model to be successful in providing modularity and flexibility to the development of models and of being capable of producing very suitable BDI-agent implementations for the modelled business processes.

Chapter 8

Obtained Results

After having reviewed the state of the art in the area of agents and Web Services integration and modelling of interaction protocols, we proposed our solution for these problems and showed how these perform using some examples from the multi-agents community and the industry.

The present section will explain in the detail the results achieved during our work:

- Study of Web Services, FIPA agent platforms and a survey of contributions around the idea of integrating these two.
- Study of Business process modelling and different approaches found in literature.
- An integrating solution for FIPA Agents and Web Services was proposed based on communication standards mapping.
- A consolidating, modular and reusable meta-model for interaction protocols was defined
- Executable implementations that make use of the Web Services and FIPA-Agents integration, can be produced automatically out of models.

These results will be explained in detail in the following sections. They have been organized in the two areas of work, Agents and Web Services integration first, and modelling of Declarative Protocols second. Finally the results achieved by combining these two will be described.

8.1 Web Services and FIPA-Agents integration

In the first part of our work we showed how the problem of integrating (FIPA) agents and Web Services can be solved. In opposition to other proposals, our integration is based on the messaging framework of FIPA agents.

We found out that there were not that many differences between both communication architectures, since both are services oriented architectures. We took advantage of this synergy and proposed an integrating architecture and an implementation that tried, as far as possible, to stay transparent.

Next, the various aspects and results will be described in detail.

8.1.1 Message properties mapping

Integration was achieved based on the integration of both communication standards: FIPA for agents and WS-Addressing for Web Services. Both of these specifications define a structure for messages, their *envelope*.

A mapping of the properties that compose each of these envelopes was proposed in Section 4.2. It has the property of being *bijective*, the mapping implementation produces semantically the same message envelope after translating from one standard to the other and back.

8.1.2 Compliance

This mapping enables compliance of messages in both scenarios. The messages generated out of FIPA messages are compliant with WS-Addressing standards and vice-versa. WS-Addressing based conventional messages (not coming from an agent) are successfully handled by FIPA agent platforms.

8.1.3 Mutual Accessibility

Since the FIPA agent platform is capable of sending WS-Addressing compliant messages and receiving Web Services messages and introducing them to the internal message transport service, the resources, agents and Web Services, are made accessible to each other.

Agents can reach and be reached by conventional Web Services and vice-versa. It is possible as well, to let agents communicate with other agents or Web Services with other Web Services using our proposed way of using WS-Addressing.

8.1.4 Provided Implementation: JadeWSMTS

Based on our theoretical work, the proposed mapping was implemented using Jade as FIPA agent platform and Axis2 as a Web Services tool. The implementation is called JadeWSMTS (Jade-Web Services Message Transport System) and was described in Chapter 6.1

8.1.5 Codec: FIPA Message Envelope using WS-Addressing

As part of JadeWSMTS a codec was implemented. FIPA agent platforms like Jade use modules called *codecs* to produce and parse messages. There

are codecs for message envelopes and codecs for message contents. In our case, we provided a codec for the message envelope only. This codec is the implementation of the mapping defined in Section 4.2. Some of the features of our tool are implemented also in this codec, like

- mapping of stateless and stateful communication parties
- handling of REST services
- handling of synchronous and asynchronous messaging.

8.1.6 Stateless-stateful communication

Web Services, as most Web based technology, are conceptually stateless and regarded as resources to be reached through an address. Agents, on the other hand, are stateful and individually identified. Our tool provides as solution the introduction of *anonymous* agents as representation of conventional Web Services. In Section 4.2 Agent IDs (AID) is added to the WS-Addressing End Point Reference (EPR) using the extension features provided by WS-Addressing for this purpose.

As a consequence, it is possible to let agents and Web Services interact with each other regardless of one being stateless and the other one stateful. Also, as a result of this, agents intended to be used this way, need to be adapted appropriately to harmonize with their stateless partners.

8.1.7 REST support

A common way, alternative to SOAP, for Web Services communication is REST communication, see Section 6.1. JadeWSMTS provides a way for agents to provide an agent specific address and through this same mechanism get REST messages.

The syntactical structure of messages to be used in the REST format is also provided, letting agents interact with other Web Resources using REST communication.

8.1.8 Transparent for agent Implementation

JadeWSMTS, as already mentioned, was implemented as an Message Transport Service (MTS) for Jade. It provides its own codec and fits very transparently in the framework defined by FIPA and is implemented in JADE.

As a consequence, agent implementations need close to no modifications in their implementation. Solely, the correct usage of addresses for new interaction partners has to be modified to reach new partners over the Web and an extra annotation in the FIPA-ACLMessage Envelope. Desiring to use the Web Services codec is necessary in a message for it to be transported using the Web Services infrastructure.

8.1.9 Possible Complex interaction protocols

The proposed mapping in our solution and its corresponding implementation lets complex interaction protocols be performed using Web Services standards. The technology necessary to implement parties capable of performing such conversations is provided by the multi-agent platform.

Our mapping and tools enable its use over Web Services and also the easy porting of existing implementations to this new communication mechanism. This way it is not only possible to implement complex conversations between Web Services, but also, it is possible to run existing scenarios we have using the new tools provided here. For instance, we could perform successfully complex simulated trading implementations. The conversations carried out by this use case reach scales in the magnitude of hundreds of thousands of messages. These conversations can be performed successfully, using Web Services, by using JadeWSMTS.

8.1.10 FIPA Agents and Web Services Integration (AWSI)

The FIPA Agents and Web Services Integration (AWSI) group, dedicated to the creation of new specifications in this area, gathered different approaches for integrating agents and Web Services (Greenwood et al. 2007). Our solution was proposed as a suitable communication framework.

8.2 Declarative Protocols modeling

A meta-model for declarative interaction protocols was defined and implemented as part of this thesis. It was developed as part of an extensive set of tools our department implemented in the scope of the project SHAPE (Benguria et al. 2009). Our target was to provide automation in the development of complex conversations, specially in the area of business processes. The declarative meta-model proposed here is one alternative we propose for the modeling of interaction protocols.

The results achieved will be described next and the examples of Chapter 7 will be used to illustrate the results. Particularly, one fragment of the Contract-Net use case will be used to highlight some of them. Therefore, this fragment will be described, more deeply and enhanced, in order to show the concepts being described in a more clear way.

This is a section of the Contract-Net Protocol shown in Figure 7.3. The particular part is where the Initiator makes the decision of accepting or rejecting the proposal. Its model representation is shown in Figure 8.1. It shows on top, various proposition names used to describe the elements in the protocol. On the left are the two roles involved: Initiator and Participant. The darker, rounded rectangles, labeled with circled numbers, are the State Descriptions we want to focus on in this example and the two big

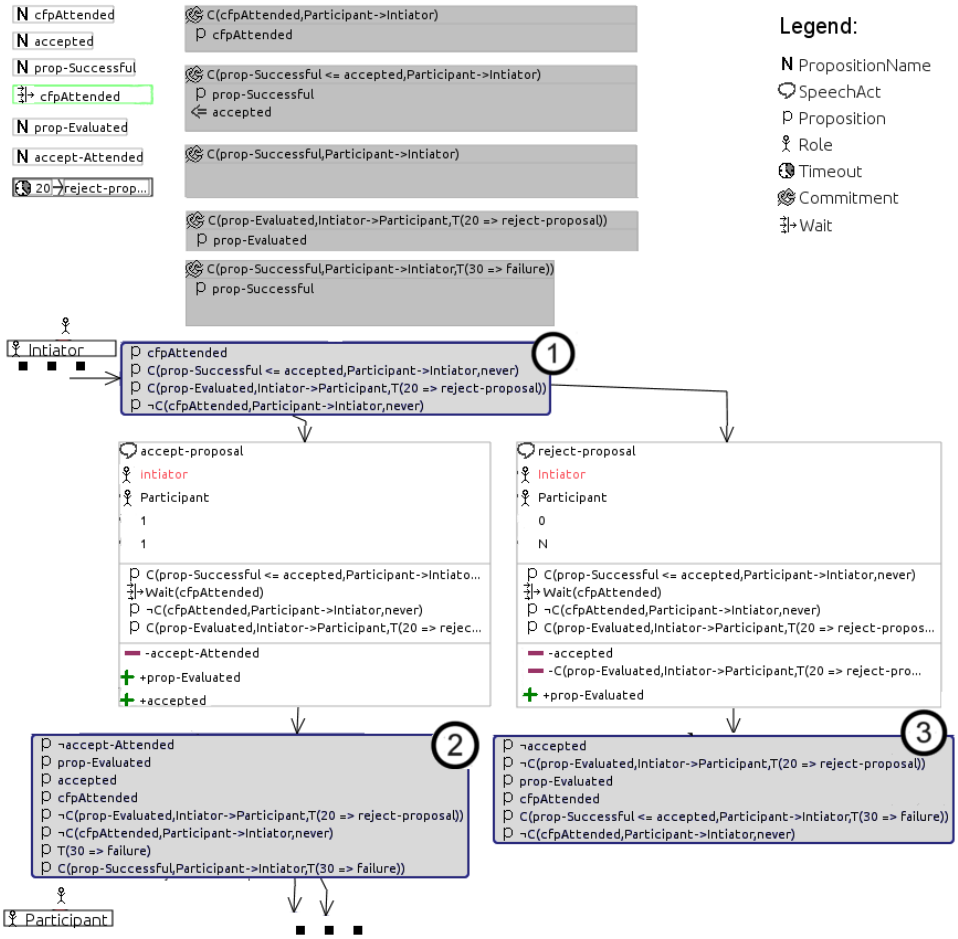


Figure 8.1: Example: Accept section of the Contract-Net Protocol

white rectangles are the two possible Speech Acts with their corresponding resulting State Descriptions bellow.

The representation using swim lanes of Figure 7.3 was changed to a different layout that focuses more on the StateDescriptions and Speech Acts, in order to highlight the concepts we want to observe better.

In the first State Description (1) one can see: that a call for proposals (cfp) has been attended (cfpAttended), that Participants are committed to making their proposal successful (prop-successful) if they are accepted, and that the Initiator is committed to evaluate the proposal of the Participant (prop-Evaluated). Finally, the Participant has been freed from the Commitment to attend the cfp, which is a consequence of bringing about cfpAttended (not seen in the figure).

Next, our results concerning declarative Protocols and their meta-model will be described.

8.2.1 Development of a meta-model for Declarative Protocols

The first and main result was the definition of a meta-model for Declarative Protocols. In it, the relations between roles, Speech Acts, and statements about the state of the conversation, called propositions, were defined in detail. Speech acts are defined as operations that change the truth value of propositions. Protocols are defined as sets of Speech Acts that produce a directed graph, similar to a finite state machine that represent the protocol model. The graph is calculated automatically and its starting and ending State Descriptions are used as connection points to establish transitions to other protocols.

8.2.2 A consolidating meta-model

The meta-model is inspired fundamentally by OWL-P (Desai et al. 2005*b*) and consolidates several concepts from different sources:

- Forward Planning
- Finite State Machines
- Speech Act theory and models
- Special proposition types specific to the subject of interaction protocols from other contributors, like Commitments; concrete definitions of widely known concepts, like Timeouts; and new contributions that improve the expressiveness of the models, like conditional propositions and Cross-Conversational Constraints.
- Interaction Protocols

8.2.3 Reduced ambiguity

In Chapter 5 the Declarative Protocols meta-model was defined formally. It is a meta-model that consolidates many concepts and contributions from others, most of them described only abstractly in their corresponding publications. The provided definitions concretely relate concepts to the rest of the meta-model. Details about these relations were defined in a concise manner, disambiguating the usage and meaning of the concepts.

8.2.4 Same meta-model for Speech Acts and Interaction Protocols

The meta-model unifies the mechanisms for defining Speech Acts and Interaction Protocols. A descriptive approach, based on propositions, is used to define Speech Acts. The combination of the defined Speech Acts produce

a graph of possible sequences in which the conversation can be carried out: the Interaction Protocol.

As a result, Interaction Protocols and Speech Acts are defined using the same semantics and removes therefore any barrier that could exist, when moving between the two levels.

8.2.5 Improved expressiveness

The fragment of a Contract-Net Interaction Protocol shown in Figure 8.1 shows how the state of the conversation is described in each State Description. This not only helps the reader get oriented, but more importantly: automatic support mechanisms can take advantage of this machine readable contents.

Declarative constructs make new conversation management techniques possible. New constructs for modeling situations specific to Interaction Protocols can be introduced into the meta-model. These constructs enhance the possibilities of expressing concepts relevant to the subject being modeled.

In our model, contributions that have proved to be relevant at the time of defining Interaction Protocols have been introduced, like Commitments and Timeouts.

Other concepts have been added, like conditional propositions and Cross-Conversational-Constraints. In the Contract-Net fragment, an example of the second one is used to force the Initiator to answer with an *accept-* or a *reject-proposal* only after all proposals are received. In the previous part of the Contract-Net protocol, all actions that respond to the *cfp* bring about cfpAttended. This proposition is used in a Cross-Conversational-Constraint (called *Wait* in the diagram, see Legend). Using this wait as precondition for the two actions shown in the fragment means that the sender of the action, Initiator, has to wait for the associated proposition (cfpAttended) to be valid, in all conversation parts of the protocol, before it performs any of them for the first time.

This illustrates that as a result of having a declarative meta-model, more details can be expressed about that model. Other constructs that add to the expressiveness of our meta-model will be mentioned in the further sections.

8.2.6 Cardinality management

Section 5.3 explained how cardinality constraints work. They are intended to manage the amount of conversations, in the scope of an Interaction Protocol enactment that can opt to perform certain constrained Speech Acts. In this way, an Interaction Protocol model can constrain how many conversations can take a certain path in the graph that represents the whole protocol.

How these constrains work can be seen in the example fragment: the Initiator is forced and limited to accept only 1 proposal, since it can perform

at least and at most once *accept-proposal*. On the contrary, *reject-proposal* is unconstrained in this aspect (0-N). These constraints allow the designer to define how many proposals can be accepted within a protocol enactment.

8.2.7 MDA-tools and Visual editor

In the same way as with the rest of the contributed tools, our department provided to SHAPE, this meta-mode was implemented using MDA techniques. A meta-model implementation and visual editing tools were created to aid the creation of models. These tools were implemented using the Eclipse Modeling Framework (EMF) and its Graphical Modelling Framework (GMF).

The editing tool was organized in two levels:

Interaction Protocol Editor

The first one lets the designer edit Speech Acts and visualizes the resulting Interaction Protocol as a directed graph. In this editor level, proposition names and roles can also be defined and used in the definition of Speech Acts.

Examples of this editor are shown in Figures 7.2, 7.3, 7.8 and 7.9.

Interaction Protocol Composition Editor

The resulting Interaction Protocols can be used to compose more complex ones using the second level editor. It visualizes the available Interaction Protocols as black-boxes annotated with their starting and ending State Descriptions. This level lets the designer decide which ending and which starting State Description to bind in order to produce composed Protocols. The editor includes a functionality to help the designer by highlighting matching starting or ending State Descriptions for selected State Descriptions.

Examples of this editor are shown in Figures 6.7, 7.6 and 7.7

8.2.8 Automatized support for model development

A visual editor that makes use of the logic defined in Chapter 5. Reduces complexity by automatically calculating the FSM graph.

The meta-model has been designed in such a way that the graph representing the protocol can be calculated out of the defined Speech Acts. This calculation has been included in the EMF implementation and supports the development of models, by providing instant feedback about the changes made to Speech Acts and their effect in the whole Interaction Protocol.

A significant amount of work, consequence of the inherent complexity of a declarative approach like ours, is taken away by the provided automatized functionality.

Apart from the automatic calculation and visualization of the graph, the meta-model implementation includes automatic modifications to constructs part of the model. This is the case of Commitments: they are modified, according to what happens in the conversation. Honored Commitments are withdrawn; Conditional Commitments are made active, as soon as the condition is brought about; Timeouts are brought about as soon as the associated Commitment is made active, etc. The example fragment shows how the Commitment of the Initiator to make prop-Evaluated true in State Description (1) is removed as soon as it is brought about by *accept-proposal* and by *reject-proposal*, since these actions fulfill the objective of the Commitment. The conditional Commitment to make prop-Successful, if accepted is made true, is converted into the corresponding active Commitment and the associated Timeout is also brought into Sate Description (2).

8.2.9 Protocol composition

The model can be used to compose protocols out of existing Interaction Protocols. As already explained in Section 8.2.7, protocols can be visualized as boxes with entry and exit points, and matching ending and starting points that can be connected to compose the protocols.

This is one of the main objectives proposed in Chapter 2 and the way modularity is supported within the meta-model.

8.2.10 Automatic generation of executable code

Using MDA-Transformations we produced automatically BDI-Agents capable of performing the business process as specified. It proves that the combinations and connections, used during modelling, produce coherent models that can be performed.

A mapping to Jadex BDI agents, a declarative architecture for MAS, has been provided. The implementation is generated using a pattern that will be explained next.

8.2.11 BDI based turn taking pattern

A plan pattern for BDI-agent implementation was presented in Section 6.2.3. It takes advantage of the declarative aspect of the meta-model. Agents evaluate protocol conformance based on the State Descriptions as specified in the protocol.

Each turn is organized separately as a set of plans. The different tasks inside each turn execution are separated into parts, each one related to a different aspect: message receiving, context management, message attending

and replying. All plans are goal driven, therefore additional plans can be added to the agent in order to improve specific tasks.

This mechanism can be used to enhance the agent implementation with domain specific concepts. The way the context the of the conversation is evaluated or the way the messages are attended can be improved, by adding or replacing these plans with plans that include domain specific reasoning.

8.3 Realization of business process using agent communication

In the example of Section 7.2 we have implemented a complex business process defined by our industrial partner Saarstahl in the scope of the project SHAPE. This same use case was used to evaluate other modeling solutions that are part of projects like SOAML and PIM4Agents.

This shows how modeling business processes, taking advantage of modularity and reuse, can be done. The complete process was designed with an Interaction Protocol model that is composed of two Interaction Protocols.

From the model an implementation was automatically generated. The generated agents are capable of interacting using Web Services standards making use of the tools provided as part of this thesis. The result is, an Interaction Protocol model used to design a complex business process and a generated implementation that performs it using agent communication over Web Services. An example of one possible interaction is shown in Figure 7.11.

8.4 Summary

Our work can be organized into two parts:

- Integration of Web Services and FIPA-Agents
- Declarative Protocol modeling and code generation

This twofold organization matches the organization of the research projects (Benguria et al. 2009) (Benguria, Larrucea, Elvester, Neple, Beardsmore & Friess 2007) in which our department participated: SOA and MDA.

Agents and Web Services integration was achieved by proposing and implementing a Message Properties mapping that is used to serialize and de-serialize FIPA ACL-Messages using Web Services standards. The implementation is called JadeWSMTS and provides an integration, it is; mutually accessible, both Web Services and Agents have access to each other; it conforms with Web Services standards, mainly WS-Addressing; and solves many other problems, like the stateful and asynchronous nature of agents against the stateless and commonly synchronous nature of Web Services. The tool

lets agents also interact using REST communication and most importantly, enables the enactment of complex conversations using Web Services.

A declarative meta-model for Interaction Protocols was defined conceptually, based on common concepts like forward planning and finite state machines. This model disambiguates and consolidates many concepts and contributions from the community in the area of interaction protocols in a single model. Concepts that improve the expressiveness of an interaction protocol model, like: Commitments, Timeouts, Cross-Conversational Constrains, Conversation Cardinality management, etc., were introduced in a concrete and precise manner the meta-model. The result is a meta-model that lets designers produce Interaction Protocols that can be composed by connecting semantically matching concrete connection points of protocols.

Implementation of models designed with our tools can be automatically generated. In the particular case of our protocol meta-model, a mapping to Jadex BDI agents was provided. This implementation explicitly used the declarative approach, by taking advantage of the declarative nature of BDI agents. Jadex agents, using the Jade extension can use JadeWSMTS to communicate using Web Services.

The end result is a set of tools that are used to model business processes as modular and reusable Interaction Protocols. Out of these Interaction Protocol models a Web Service implementation can be generated. This way, agent communication, and more precisely FIPA standards, are used to realize business processes.

The capabilities and results were shown using an industrial use case provided in the scope of SHAPE (Benguria et al. 2009) by our industrial partner, Saarstahl. This use case was modeled by composing the complex scenario using two Interaction Protocols. An implementation was generated and executed using JadeWSMTS as Web Services communication platform.

Chapter 9

Analysis and Evaluation

Tools that enable the communication of FIPA agents using Web Services standards and also tools for modeling interaction protocols using a declarative approach have been defined sofar. Based on the definitions and examples provided, the results will be analysed and evaluated, by comparing them to other tools and approaches.

The analysis will be done in two parts: first the Web Services integration will be evaluated and after that, the declarative modeling tools. How these are used together will be evaluated by referring to the examples and the industrial use case. In the last section, a comparison with contributions, described in State of the Art in Chapter 3, will be provided.

9.1 Messaging integration

9.1.1 Web Services and FIPA-Agents integration

At the beginning of our work, there was a big gap between common Web Services and the communication standards used by multi-agent systems, represented mainly by FIPA standards. Several integration approaches between the two have been described in Section 3.4.

In general, the approaches that conform to the recommendation of Agentcities have some properties in common, like:

- Not being able to perform complex conversations
- Not using Web Services standards, many of these were proposed at a time when such standards had not been established
- Being rather limited concerning scalability
- Not taking advantage of features in the FIPA model and implementations designed for this purpose: where the agent platform is to be enhanced with new communication capabilities.

Our approach is very advantageous in those areas. It is capable of performing very complex and extended interactions. It is based on established Web services standards like WS-Addressing. It is very scalable since it does not perform any replication of resources, but instead just remains the communication mechanism part of the agent platform. There are other aspects where other integration approaches have advantages, like being capable of translating services descriptions or being able to translate message contents. An in depth comparison of each of the relevant contributions is provided in Section 9.5.6.

9.1.2 A framework capable of complex conversations

The Web Services community tried several times to produce standards to define how a set of agents should interact with each other to achieve a goal. The most important proposal was Web Services Choreography Description Language WS-CDL (W3C 2005) described in Section 3.5.2. These proposals where unsuccessful, because of the increase in complexity in terms of message semantics and new concepts that have to be taken into account, like social commitments, turn taking, conversations management, etc. WS-CDL could only model conversations between two individual agents.

Paradoxically, most of the integration solutions done by the multi-agents community were restrained to the RPC nature of Web Services instead of introducing useful concepts used in the FIPA architecture.

Later improvements in Web Services messaging for performing complex conversations (WS-Addressing) allow a smoother Web Services compliance for a FIPA grounding that goes beyond request-response situations. These improvements in Web Services manifest movement towards a technology more similar to multi-agent systems. This implies that problems in Web Services will become more similar to the ones studied for agents. For instance, FIPA specifications, can handle situations very well WS-CDL intended to solve. Our solution not only incorporates the most recent standards for Web Services, like WS-Addressing, but it also provides a model to describe all these fundamental concepts necessary for a complete description of how a complex conversation is to be performed.

Other solutions for integrating Agents and Web Services did not serve the ultimate goal of performing complex conversations that go beyond request-response. The majority of them, shown in Section 3.4, follow the Gateway approach. This, in fact, is not a transparent integration, but instead, the introduction of an adapter module that serves as a translator. The adapter pattern in Web Services to integrate heterogeneous systems is primarily addressed to legacy systems (Barry 2003). Our approach, instead, makes use of actual WS-Standards to realize FIPA messages. Agent Platforms are per se SOAs and share similar mechanisms as a Web Services architecture. As shown in Section 4.1, both architectures share too many similarities to jus-

tify an adapter and even less a translating module. Our approach is simpler, more versatile, and effective. It is based on features that were introduced already in FIPA standards for the purpose of being able to represent FIPA messages using different serializations. These were used to introduce a new way to represent messages using Web Services inside the FIPA agent platform. The direct advantages are transparency for the agent implementations, freedom to perform complex conversations as already usual in agent platforms and better Web Services compliance.

9.1.3 Web Services as grounding for FIPA specifications

Web Services are a good grounding for FIPA specifications; they provide a widely adopted communication infrastructure and are focused on the representation of information and its transport; leaving free room for application semantics. FIPA is specified in a way that complements well, because it lets semantics be grounded using different implementations.

Apart from that, messaging works very similarly in both cases. Our integration approach takes advantage of the synergies, explained in Section 4.1, to integrate both scenarios in a transparent way.

Simplicity has been an advantage for the acceptance of Web Services, which would be a very helpful feature for multi-agent systems. At the same time, these can help in the further adoption of Web Services in more complex scenarios.

A transparent integration, like our communication architecture, provides a platform to transfer concepts like dialog games, social commitments, security, negotiation, FIPA models, etc. to Web Services. The multi-agents community has studied these issues in detail and gained important experience. One of the most important projects that provided input was Agentcities (Agentcities 2002), a project in which a worldwide network of agents was created.

Nevertheless, the agent community has had trouble integrating with Web Services. For instance: Jade, the most popular and widely used FIPA-compliant agent platform, made some effort in this direction. It provides a http Message Transport Protocol (MTP) and XML encoding specifications for its messaging. Still, these means are not sufficient for true Web Services interoperability: the http MTP and the XML schema for the codec do not follow any standard, but that of FIPA. Web Services standards like Simple Object Access Protocol (SOAP) and WS-Addressing have no chance to be used in this context.

9.1.4 A transparent integration

Since JadeWSMTS is integrated into the agent platform as a Messaging Transport Service, it can be used with the mechanisms for communication

already provided in the platform. This makes it very simple for agents to use the provided Web Services communication capabilities. The same ACL-Messages are used by agents to communicate using Web Services.

From the perspective of conventional Web Services, the messages issued by agents follow common Web Services standards. Therefore, these messages are treated by Web Services implementations and infrastructure as any other Web Services message.

This transparency is also the result of avoiding a translation of message content. The reason for this was that contents is domain specific and therefore left for a different layer in the communication stack to deal with it. The way to handle communication in a FIPA and in a Web Services architecture is by separating envelope and contents handling, leaving the second one to application modules.

9.1.5 Coping with technical and conceptual differences

Our integrating architecture takes care of some technical issues that arise when agents and conventional Web Services have to work together. The first case is, how to handle the *statefulness* of agents which are identified uniquely throughout their lifetime. In the mapping exposed in section 4.2, this was solved by adding the agent ID as an extra parameter to the WS-Addressing Endpoint Reference (EPR), a common practice in similar situations when using WS-Addressing. The difference in the mechanism for message correlation was also solved this way. Another way of solving the identity mechanism of agents would be to assign them a unique address that will belong to them during their complete existence. This would be a simpler solution for cases where agents do not *move* causing the address to be invalid. It could be enhanced also with a mechanism for message forwarding so that the agent can still be found under the same address even after moving away. All these possibilities remain open using our tool, even though the mechanism for using agent IDs is preferred. Agents communicating with conventional Web Services do have to be able to interact with parties that have no ID as used by agents. Our tool introduces the agent name "ANONYMOUS" to represent such agents. This way, at least, participants that prefer not to be uniquely identified, but only be referred to by their address (like conventional Web Services), are well represented.

9.1.6 Coping with parties with different reasoning power

In general, the strong variation in complexity manifested by participants in such an integrated architecture requires the introduction of some assumptions. Very simple request-response supporting participants would have to interact with complex agents that cope with a wide variety of message types and that are not deterministic in their behaviour. The messaging mech-

anism has some requirements that will not always be supported by very simple participants, for instance, the inclusion of the type of protocol the messages are part of. Very simple third party services will not mention such information. In this case, the assumption would be that when a protocol is not mentioned, a simple request-response protocol is implied.

One approach for supporting complex conversations with partners that possess little reasoning power was proposed in (Ardissono, Goy & Petrone 2003): an orchestrator service that performs the reasoning about the dialog and guides the participants with the Speech Acts available to proceed in the dialog, hindering the autonomy of agents. EPRs use their *meta-model* field to describe the service interface of an Endpoint. This is a feature an agent can use to provide a detailed description of how an answer is expected, an alternative that lets agents interact with entities of less reasoning power, but without compromising autonomy. Using REST communication could serve the same purpose.

9.1.7 Using existing Web Service Tools

Our implementation, called JadeWSMTS, provides some advantages like the delegation of message transport to Axis2 (AXIS2 2006) (a Web Services tool). It allows JadeWSMTS to stay up-to-date with less effort and simplifying Web Services compliance. In fact, JadeWSMTS provides, at the moment, the most modern FIPA ACL message representation using Web Services standards. This takes better advantage of Web Services messaging infrastructure, something useful when implementing agents that should work in a SOA. In contrast, (Sonntag 2006) proposed specific SOAP headers to implement a service mediator in charge of forwarding requests to agents, performing the actual task, JadeWSMTS would have simplified this significantly: WS-Addressing already provides the required headers which are used accordingly, based on the information in the FIPA ACLMessages. Any changes in WS-Addressing, or any other related WS-* specification, will be taken care of by Axis2.

JadeWSMTS is implemented, as any other MTS, inside Jade, which is a cleaner integration technique than using an agent or a gateway for providing such a service. Having such a natural integration inside the platform allows a very straightforward and less intrusive integration of agents not originally implemented to make use Web Services. The transparent communication integration in this tool allows agents to use UDDI. At the same time, services outside the platform can use the DF, which avoids the overhead of replications and extra translations used in some of the gateways like in (Nguyen & Kowalczyk 2005). Even though the DF can provide a Web Service interface to the outside, this feature is not expected to be used frequently by conventional Web Services.

9.1.8 FIPA Agents and Web Services Integration (AWSI) Group

Our Web Services integration solution was included in the FIPA agent and Web Services Integration (FIPA-AWSI) recommendation as a suitable communication framework for Agents and Web Services. This recommendation reflects the dominating opinion of the multi-agent community concerning integration of agents and Web Services.

FIPA-AWSI went beyond the simple proposal of bridging communication between agents and Web Services and included a proposal for Declarative Protocols called OWL-P (Desai et al. 2005*b*). This recommendation served as a foundation for the Declarative Protocol model that we provide. Many concepts and ideas proposed in OWL-P, like the importance of a declarative model to achieve flexibility and modularization capabilities or the importance of including Commitments as a fundamental semantic construct, were included in our approach.

9.2 Declarative Protocol modelling approach

In the scope of PIM4Agents (Hahn, Madrigal Mora & Fischer 2009), our department has provided a comprehensive set of tools for modeling multi-agent systems. Many aspects, from the definition of abstract interactions between roles to the concrete deployment details, can be modeled. Out of these models, executable code can be generated.

The set of tools includes modeling tools for interaction protocols which follow the conventional structured way for modeling. The approach provided in this thesis, offers a different way of modeling: declaratively. In this section, the advantages and reasons in favor of such an approach will be discussed, along with some of the trade-offs and disadvantages that come along.

9.2.1 Reasons for a declarative approach

One of the main ideas pursued by the multi-agent systems community is to have a library of general purpose protocols that can be used, by combining them, to create solutions for concrete situations.

FIPA Interaction Protocols Library is the most relevant specification of such a library. Still, this specification does not show how protocols can be combined. As part of the proposition of the FIPA AWSI Group, an approach using a declarative model was proposed: OWL-P. Based on this idea, our declarative meta-model was designed and implemented.

In order to have a library of protocols that can be used, by combining them, to produce complex interactions that serve specific purposes, a way of managing protocols and representing their connection points is necessary. A

declarative model has explicit descriptions that help in, among other things, the management of the library and support for creating valid combinations. Some of the advantages achieved by our declarative meta-model will be discussed next.

Concrete definition of constructs

A declarative approach provides a concrete representation of the meaning of the structures defined in the protocol model. These contents support different operations on protocols, like matching and composition.

Connection points, in our case, starting and ending State Descriptions, can be compared, by looking at their contents, to choose which can be connected. This information, represented inside the model, enables reasoning about the concepts, to decide how to establish connections.

Our proposed meta-model provides the mechanisms, using proposition names and operations, to define constructs used in interaction protocols models by specifying:

- The internal structure of the construct
- The relation to the other constructs
- How they work inside a protocol model

Deliberation about concepts and constructs

Even though the present work does not cover automated reasoning about propositions and State Descriptions, it leaves open the possibility to analyse, compare and make decisions based on them. This information is very useful for other areas of multi-agent systems, like goal oriented design or dialogue games.

Reasoning to make decisions based on the information provided by State Descriptions and Speech Acts can be implemented using our meta-model. This idea has been already proposed and discussed by Singh (Mallya & Singh 2006*b*). Our objective is to improve the management and coherence of protocols by using a declarative approach and therefore automated reasoning is out of our scope.

Automation and support in design of models

After exceeding a certain amount of available interaction protocols, the task of selection and combination becomes very hard. Management possibilities in the existing modeling techniques are scarce.

A declarative approach lets the model represent, in a machine readable manner, the meaning of constructs and protocols. We have taken advantage of this and provided supporting tools for the designer during the process of

developing Interaction Protocols. A machine readable representation of the concepts opens the possibility to implement supporting tools that automate part of the work and help designers.

In our case, many of the details involved in the domain of interaction protocols were formally defined in detail and were implemented subsequently in a meta-model. Many tasks, like the generation of a graph representing the Interaction Protocol or the comparison of connection points, were achieved thanks to the use of a declarative approach.

Flexibility

Our composition approach is different in that it lets protocols have many starting and ending State Descriptions for composition. For instance, the industrial use case example in Figure 7.6 uses a protocol called "Order" to model the interaction between clients, sales, and production planning. This protocol has several connection points, one main entry point and one pursued exit point, but several other entrance and exit points. These many connection points allow, on one side; to plug in different ways of solving a problem, similar to the goal of OWL-P, reducing dramatically *coupling* in the model. In the example in Section 7.2.4, this is the case with the cascading calls to QueryIf protocols, they are used by *planning* to find out if the production of the order can be confirmed. On the other side, the connection points are used to handle all the possible outcomes of the protocol. These two features translate into enhanced flexibility, not only can the protocol be combined to produce big complex interaction protocols, but also, the way agents solve situations together with other agents, can be modified with less intervention to other parts of the model.

9.2.2 Modularity

Our tool improves modularity and makes mechanical support possible, because of the way protocols are represented: protocols can be compared by looking at their entry and exit points to tell which of them can be combined as modules of a bigger composed protocol. This combination process is supported by automatically comparing possible matches of State Descriptions.

Our meta-model represents interaction protocols as modules with connection points that are annotated with their semantic. These annotations enable a better management of modules and an explicit binding of concepts. In opposite to current possibilities provided by other techniques, where combination of protocols is left completely to the intuition and reasoning of the human designer, our tool takes over part of this work.

9.2.3 Difficulties of a declarative approach

Complexity during model development

Declarative models can easily get too complex, especially in case, since the complete model and the sequencing of actions has to be calculated out of the definition of Speech Acts. Our modeling tools takes away some of the burden by automatically calculating a graphical representation.

Controlling unpredictability

During the construction of Interaction Protocol models, the user realizes how easily a model turns into an big maze of possible paths for conversations. This can only be controlled using constraints as preconditions for Speech Acts. It is an expected behavior of protocol models: the less constrained a protocol is, the more unpredictable it gets.

Unpredictability in our models can be recognized by the relation between the amount of paths and State Descriptions and Speech Acts. There are some rules added to the meta-model that help moderate these explosions of information, like the principle of effectiveness (León Soto 2009). It states that no resulting State Description of a Speech Act is added to the graph, if they are identical to the enabling State Description, in the sense that there is no effect from performing the Speech Act.

More rules like this principle can be added to generate more sensible diagrams, but it remains a decision of the designer to define the amount of freedom desired for an interaction protocol.

9.2.4 Modeling editor and its usability

In our formal definition of a declarative meta-model for Interaction Protocols, many constructs necessary for the definition of protocols were defined in detail and many more can be added. Also, typical structures and usages exist in the models created with our tools, like parallel conversations and the synchronization of them, decision points, etc.

Our visual editor produces very modest graphical representations of the model. Many of these concepts and constructs could be represented in a more intuitive way and some recurrent tasks could also be automatized. This issue is discussed further in Chapter 10: Future Work.

9.2.5 A unified meta-model

FIPA has provided the most well known set of protocol specifications for autonomous agents in the form of a set of Agent UML models. The idea behind these protocols was to the provide a set of basic protocols that can be used in combination with model complex conversations.

The way these protocols can be combined is left to the intuition and reasoning of the human designer. There is no link between the definition of Speech Acts and interaction protocols in FIPA. Speech Acts are defined in a very mentalistic manner, as it is done in dialogue games (McBurney & Parsons 2002). Such an approach takes the view of the performer of the Speech Act. Although this is useful for the agents to reason about it, it is useless for the definition of interaction protocols, because these are seen from a global perspective. In FIPA, the design of protocols relies on human intuition and reasoning to interpret the protocol diagrams and more over, to come up with plausible combinations.

Our meta-model breaks this barrier and uses the same semantics for the definition of Speech Acts and Interaction Protocols bringing them together in the same meta-model.

9.2.6 Disambiguation

Our meta-model was defined formally and consolidates many ideas necessary for the definition of Interaction Protocols. It makes use of widely known techniques like forward planning and finite state machines. Our formal definition describes concretely, how all these ideas are put together. Many of them were described in publications in a rather abstract manner, like Commitments; or are understood differently by different members of the community of multi-agent systems, like Timeouts. These ambiguities are clarified by our model definition and this is a fundamental step towards an implementation like the one provided here.

9.2.7 New constructs for Interaction Protocols

In our meta-model, constructs were defined to improve the representative power of the models. For that purpose, some constructs were created, like Conditional Propositions and Cross-Conversational Constraints, or defined in more detail, like Timeouts or Commitments.

Conditional propositions

These are helpful to represent situations where a participant makes a decision, but manifests it using the same Speech Act. This proposition produces two possible outcomes to show how the protocol goes on, depending on the state of a condition. This proposition can be frequently used in situations where the course of a conversation will be lead by a condition in the *contents* of a message.

Cross-Conversational Constraints

This feature serves as a connection between conversations that run parallel by synchronizing them and also specifying precisely what synchronisation means. In the case of the Contract-Net example of Section 7.1 and visualized in detail in Figure 8.1, the meaning of being synchronized is: to have attended the *cfp*. This construct makes the meta-model capable of specifying how each conversation is managed in complete parallelism and how they are related to each other inside a protocol, an aspect that is normally not clear in other modeling techniques.

Timeouts

In our meta-model, Timeouts are defined as a time countdown at the end of which, involved participants assume that a certain Speech Act has been performed. The relation between the Timeout and a Speech Act gives the timeout a meaning. It specifies what will happen to the conversation by using a Speech Act to represent the changes. This way, the participants expecting a reaction from other participants can update the state of the conversations after the time for the reaction ran out. At the same time, participants know the consequences of letting a Timeout run out of time.

Commitments

Commitments, as explained in Section 5.4.2, were mainly defined and implemented as proposed by Singh and his group. We have added the component of a Timeout to give the Commitment a time limit for it to be honored. In consequence, Commitments gain also a definition of the consequences when they are not honored.

New constructs

Using Declarative Protocols, a more detailed description of what happens throughout a protocol is achieved. More concepts, relevant in the context of an interaction protocol, can be introduced in the model, taking the model beyond a simple description of action sequence and turn taking.

9.2.8 Concrete and detailed meta-model implementation

Using MDA techniques in the area of multi-agent systems is relatively new and therefore, there are no known implementations of this kind of meta-models. PIM4Agents (Hahn, Madrigal Mora & Fischer 2009) provide such meta-model implementations and code generation tools based on them.

Our protocol modeling tool is the first one implemented and capable of producing concrete and usable declarative interaction protocol models. It is

also the only meta-model that aggregates different theoretical contributions and concepts from the multi-agent systems community.

9.2.9 Automatically generated code

In the same way as it was done with meta-models in PIM4Agents, source code can be automatically generated by tools part of the present work. In the specific case of Declarative Protocol models, the generated code is intended to be used by the messaging tools produced in the first part of our work. Therefore Jadex BDI-Agent capabilities are generated, which can be used by agents that perform the roles.

Declarative models help making better BDI-Implementations

In traditional automatically generated implementations of protocols, roles are implemented in single blocks of code. This is the case in jade work-flows and most of our transformations in PIM4Agents, SHAPE and ATHENA projects. This does not reflect the flexible and multi-directional nature of conversations.

In PIM4Agents, some BDI-based protocols implementations were generated from the conventional structured model. These implementations condensed the complete protocol inside a single plan-body, representing the whole role implementation using a single sequence of code. This has as side-effect the misuse of BDI's advantages because it hinders BDI-scheduling.

The declarative models can be more easily transformed to better BDI implementations. The preconditions of Speech Acts and the various State Descriptions can be used by BDI-agents to manage the conversations. The propositions can be managed inside the belief-set of the agents. Belief-sets represent the state of the conversation similarly to State Descriptions. Speech Acts are implemented as plans with conditions that match the pre-conditions in the meta-model. As a result, instead of having the whole interaction protocol inside a single plan, the implementation is distributed over a series of plans, each of which represents a Speech Act. This enables BDI-scheduling and reasoning about which action to take; every time the agent has its turn in the conversation. This produces implementations that are more flexible, reusable and scalable, and by giving the BDI-scheduling a predominant role, it improves the implementations with its favorable properties.

Turn taking pattern

For the implementation of the BDI-Agents, a library of routines was implemented. This library implements a pattern defined to handle each turn of an agent participating in a conversation.

The routine implements this using a set of plans that handle the different tasks involved: it starts by fetching and updating the context corresponding to the conversation the message is part of. After that, by attending the message and performing the domain specific work and finally, by deciding how to reply and generate the response.

This pattern enables the BDI-scheduler to take part in each turn and, more importantly, makes it easier to extend and improve the implementation of the participant by adding other plans. This way, changes can be made to the agent without having to modify existing code. This is done by providing alternative plans and assigning them priority values. Adding plans helps to improve the way agents solve each turn since having alternative plans for the each task provides multiple ways of coping with more varied situations. In cases where some plan may fail, other plans may work, making it a more robust and versatile implementation.

9.2.10 Open questions about the fundamental concepts of the meta-model

About propositions

Propositions are a fundamental piece of the whole concept of Declarative Protocols. Based on these, the chance of performing a Speech Act is calculated. Our models are seen from a global perspective and therefore each State Description is clearly known.

One question that arises is: when the time comes that an agent performs a role, how is it going to be capable of knowing the truth value of the various propositions. In principle, and taking into account that we are discussing a scenario where only communication plays a role, it can only be aware in two ways:

- by performing Speech Acts that bring about or deny propositions
- by receiving Speech Acts that bring about or deny propositions

Seeing it this way, an agent can only be sure of propositions stated directly by the Speech Acts it performs or receives. This restricts the model of the role to perform Speech Acts with preconditions that are stated by either of these two ways.

A proposition can be brought about by Speech Acts performed by a third role that does not communicate directly with a role that uses the proposition as precondition. One possibility to clarify this situation is to enable a role to assume, in a transitive manner, a proposition that is necessary in interactions that provoke Speech Acts targeted at it.

This has been the mechanism chosen in our implementation of roles. Every time a message is received, its corresponding enabled State Description is assumed.

Moreover, there could be other mechanisms on the look out for the truth value of a proposition. Propositions will frequently represent visible properties of the environment, letting agents be aware of their truth value by directly observing it or by observing the contents of the message.

These procedures are out of the scope of our work, since we limit ourselves to the domain independent level. Nevertheless, the generated BDI implementation and its pattern are structured in such a way that improvements and precision enhancements can be introduced into the plans in charge of updating the internal state of the conversation of the agent and in those in charge of attending the messages and interpreting the contents.

Creating a general purpose library of protocols

Our declarative meta-model is targeted at providing tools and constructs that help create a library of protocols that can be used, by combining them, to model complex protocols that solve specific collective needs of a set of participants. In our examples, we have shown that our meta-model can help in this task.

At this point, a question that arises is: what does such a general purpose library looks like? As mentioned already in Section 3.2.2, FIPA has already provided a library of interaction protocols. Open questions still are: are these the right protocols to have in such a library? are they all necessary? can these protocols be organized or classified? However, looking at it from the perspective of what we have achieved up to now, our first question would be, what do such protocols,described using our meta-model, look like?

Taking into account the compromise between relaxing a protocol and its predictability, a general purposed protocol will be usable as far as it constraints let it be used in several situations. This forces protocols to be as relaxed as possible, which is, as our modeling editors show, a hard task to accomplish. Some steps that can be taken in this direction will be discussed in future work in Chapter 10.

9.3 Findings gathered in the Use Cases

As part of our work, some use cases have been implemented. The insights gained from this work will be presented and discussed next.

9.3.1 Reuse of models

By looking at the model in the industrial example of Section 7.2.4 one can see that the complex business process was modeled using two types of protocols: order and queryIf. The models produced by us reuses sub-models more than other models produced for this use cases, representing a significant reduction in work and better consistency.

9.3.2 Manageability and flexibility

The multiple connection points and modules that compose the complete model for the business process makes the model very manageable in the sense of allowing easier change or replacement of parts. At the same time, this leads to increased flexibility.

9.3.3 Relationship between abstract and domain specific models

The compromises that have to be taken to reuse models are palpable in the way queryIf is used in the industrial example of Section 7.2.4. The way it is used does not match satisfactorily its name. In the first part of the protocol, it is indeed used to query each phase in the production. But in the second part, it is used to inform them to produce it, providing a mechanism to inform of the sudden impossibility of producing it. In the second half, the task is not being queried, but instructed. Still the queryIf protocol was suitable because of its matching structure to request the offered service.

Possibly, a better name for the protocol and some of the propositions would make the model more intuitive and coherent. This is a subject that is closely related to what was discussed in Section 9.2.10, concerning a general purpose library of protocols.

9.3.4 Web services performing Business Processes

Using our model for the business process and implementing it by using the generated code provided a suitable Web Services implementation of the Business Process. A long-lasting and complex conversation could be enacted by the participants using standard-compliant Web Services.

The use case showed that techniques used in multi-agent systems and the specifications provided by FIPA can be used together and are suitable for the implementation of business processes using Web Services.

9.4 Overall evaluation

Modeling and implementing the use cases was not a hard task compared to other modeling techniques. In fact, having the possibility of automatically generating runnable code, makes the task of creating and implementing a complex conversation easier.

Trying to solve a problem using declarative frameworks will inevitably bring along an increased complexity. The implementation of the meta-model was correspondingly more difficult than other approaches, but the usage of well known techniques, like forward planning and finite state machines,

helped produce visual editors that support and alleviate the development of such intricate models.

The result is, a set of tools that reduce the overhead of a declarative approach, reducing their creation effort to a similar level as other mechanisms and taking advantage of a declarative approach, in our case: modularity, reuse, flexibility and better BDI implementations.

9.5 Comparison with other approaches

Our work is one of the first to approach the problem of modelling business processes using a declarative meta-model. The multi-agent community agrees in that a declarative approach brings along many difficulties that make it a less attractive option. Due to its complexity, results tend to include many details. However, at the same time, these contents, once present in a model, come in very handy, since it allows more automatized and sophisticated tools.

Next, keeping in mind our objective of realizing business processes between autonomous participants, we will compare our tools with the most prominent options available, discussed in the Related Work.

9.5.1 BPEL

BPEL, as a language, is capable of describing business processes for Web Services, but it is intended to produce models that will be executed by an orchestrating machine. In this sense, a meta-model is closer to programming languages than for describing interaction protocols between autonomous participants. It does have the advantage of being used in a global perspective, but it stays at this global perspective and it never leaves it, even during execution.

We are more convinced that at the modeling level, a global perspective is very advantageous for defining the business process to be left behind as the development process evolves towards the implementation of the participants, which is intended to be done independently of each other.

An interaction protocol, as in our case, is to be used as a contract between the participants. Implementation is to be left free.

The most successful solution for implementing composed Web Services was WS-BPEL(Andrews & et. al. 2003). It provides, among other components, a modelling language to describe how different Web Services are to be put together. This is done by *orchestration*, where a WS-BPEL engine interprets the described model and performs the service calls as defined in the model. This is very different than what is being pursued by us, namely a *choreography*, as explained in Section 1.3.

9.5.2 BPMN and Jade Work-flows

We have collected the two in a single section, because they have certain properties in common. BPMN is the most prominent modelling tool used for business processes.

In this case there are two main differences that are important to highlight:

- Modularity: even though BPMN and similar meta-models are very suitable for describing processes, they are not that suitable when it comes to reuse and modularity, specially, when a combination of different processes as part of a business process are to be taken into account.

  Having a declarative approach has the direct advantage of not only being able to describe the modules with the same constructs used to model them, but also to reason about them and aid in the model development and management. Declarativeness brings unconditionally complexity along, therefore supporting automatic reasoning is necessary in order to be productive using these models, but at the same time, it levers the possibilities for these automatic tools significantly in comparison to structure based methods like in BPMN.

- Perspective: BPMN and Jade Work-flows provide constructs very suitable for defining the internal structure of the participants in a business process. The communication between the participants has, in fact, a slightly secondary role and the behaviour of the system as a whole is to be understood from the perspective of internal behaviours of the participants.

  Their success and wide acceptance, specially in the case of BPMN, could probably be a result of this. It provides a meta-model very suitable for defining these internal processes, which, as a matter of fact, is of much interest for the designers of a business process that usually represent each of the roles that will be involved in the process. For them, it is of great interest; what they will do and want to indicate inside their process, and when and how they will communicate.

  Our approach is very clear in starting from a global perspective: how the collection of participants is to interact and work as a system. Internal details are left completely out of scope and intended to be approached later by each participant. We call this process to *project* the interaction protocol to the specific role. The advantage of this approach is that; it is less prone to include tendencies resulting from internal details of a participant, helping to keep the global model more pure, and respects the autonomy of the participants. At the same

time, this projecting step, can be supported with automatic tools, in the same way we produced executable agents.

9.5.3 OWL-P

Our meta-model was significantly inspired by OWL-P. They proposed a declarative approach and included Commitments as a key concept in the ontology and the overall logic, because it is very useful for composition.

We have taken these conceptual principles and enlarged them in a comprehensive meta-model. We added more concepts we think are also necessary and even extended the Commitments by adding Timeouts to them, to have a more pragmatic representation of the effects of not attending them.

Out of these ideas, a complete MDA-tool has been developed, capable not only of modelling processes with an automatic supporting tool, but also of generating executable code out of these models.

9.5.4 AMOEBA

Amoeba is a methodology that came from the same sources as OWL-P. Even so, it has not taken advantage of the details a model can have when it is done declaratively. In the methodology, some steps have been left somewhat ambiguous, like sequencing of messages and why they form protocols in the form they do. These aspects, instead of being concretely grounded in the same model, are in fact organized solely by the human reasoning designer.

A fundamental conclusion of our work is instead that a declarative approach can only be of benefit if it provides tools that automatize parts of the reasoning necessary to develop and manage models. Otherwise it will not be useful once realistic scenarios come into play.

9.5.5 FIPA and Dialogue Games

FIPA provided a somewhat comprehensive library of interaction protocols. These are defined using AgentUML, a diagramming technique with many concepts relevant to the agent community. As the first example in Section 7.1 shows, our meta-model can be used to represent these protocols declaratively. This would add the benefit of being able to manage this library with more automatic support provided by tools like those we have provided.

FIPA has also a declarative definition of Speech Acts which is done as in Dialogue Games. These definitions are based on the internal concepts of the agents, like their beliefs and intentions. Therefore, they are not helpful in case one wants to define interaction protocols based on this same semantics. Interaction protocols, like in FIPA, are declared from a global perspective. The internal definitions of its participants must be transparent at this level.

9.5.6 Agentcities, WS2Jade, AgentWeb Gateway

In principle, the different strategies share the same idea of a wrapper or adapter module. This is the recommended way to integrate heterogeneous systems to a Web Services architecture (Barry 2003). Web Services used to lack support for complex conversations and accordingly, integration with agents was done using wrappers(Jennings 2001). The experience gained in the Agentcities project (Agentcities 2002) proposed enabling interoperability using a gateway (Agentcities Web Services Working Group 2002) for the interaction of services and agents. Several solutions have adopted the Gateway approach (Greenwood & Calisti 2004), (Curbera, Khalaf, Mukhi, Tai & Weerawarana 2003), (Singh & Huns 2005). Most of them focused on simple Web services request-responses conversations not enabling more complex interactions. Even so, significant results were achieved in the mapping of description (Greenwood & Calisti 2004) and the complex semantics (Nguyen & Kowalczyk 2005), (Greenwood, Nagy & Calisti 2005) which are areas where approaches tend to be different to those proposed by FIPA.

The original integration approaches of the multi-agent community followed the recommendations from agent Agentcities and the use of adapters. At that time, that was the most suitable solution and was used by most of the integration tools at the time. But soon after that new Web Services standards, like WS-Addressing, appeared and a more transparent and integrating solution like ours was possible.

Some of these solutions have the problem of replicating the complete Web Services world inside the agent platform. This means that all external Web Service resources that are to be used by agents inside the agent platform will be replicated, which implies serious scalability issues.

On the other hand, we have never approached the integration of agent descriptions and Web Services descriptions like some of the other solutions did. We regarded the description and discovery issue, in our scope, as one more communication phenomenon.

Significant work in the area of Agents and Web Services description and discovery has been done by other collaborators of our laboratory .

9.5.7 Rosetta-Net

Rosetta-Net provided a big library of small interaction protocols. From the modelling point of view there is not much to gain from this project since all these protocols are request-response protocols. On the other hand, it proved the advantages and usefulness of using interaction protocols libraries.

9.5.8 WS-CDL

Web Services Choreography Description Language was never adopted as a community wide standard. Its main flaw is that it is not possible to model

and manage the different cardinalities of interaction partners. Concepts like Cross-Conversational Constraints, something fundamental for interaction protocols, is not possible using this standard.

It is also a very low level modelling approach that eventually would have issues concerning manageability, modularization and reuse.

9.5.9 SOAML, UML2 enhancements and AMP proposal

Our work was also used in European Projects like, ATHENA, SHAPE and COIN. Therefore many concepts, like roles and bindings, were adopted from these modelling techniques. Similarly as with PIM4Agents, our approach differentiates itself by being declarative. With our models we can manage and combine interaction protocols in ways that are not possible with the procedural or structured techniques of these other works.

9.5.10 PIM4Agents

Our modelling tools are part of PIM4Agents and represent an alternative mechanism for the specification of interaction protocols. We can compare a declarative approach and a structural approach very well, because both are based on the same meta-models.

The main advantage of the structural approach is its manageability. The interaction protocol in PIM4Agents is very well interlaced with the other viewpoints. We can recognize the amount of complexity that differentiate both situations. Without the support of the automatic diagram editors, a declarative approach would have been almost impossible, which makes it much more difficult to implement.

On the other hand, the declarative approach makes the meta-model more flexible and less arbitrary. New concepts are easy to integrate by defining it using the clear semantics of our meta-model. Concepts like Cross-Conversational Constraints and Commitments were are very well integrated in the meta-model which, at the same time, is not that clear or possible using the structured approach.

Also the explicit modularity that is achieved using a declarative approach provides great advantages in comparison to the structured approach, which is not as successful in concerning this.

Apart from that, as an extra bonus, the declarative approach is easier to map to a BDI model, something very interesting for our department. BDI-agents are also declarative; they have descriptions of the world and by reasoning about them, they choose their next action. Goals and Propositions are closely related and help to produce BDI execution models that follow the Goal-Plan mechanisms better. With PIM4Agents it has been very difficult to produce executable code that is not, in principle, a single portion of code

in a traditional programming language. These code fragments impede BDI reasoning significantly.

9.6 Summary

The proposed solution for message integration in a Web Services scenario for agent platforms is the only one capable of integrating agents and Web Services transparently. This is achieved such that agents can interact in complex conversations in the same way as in agent platforms. At the same time there is also no visible difference, from the outside, between autonomous agents and other implementations behind the Web Services interfaces. This opened an opportunity for developing complex conversations using Web Services, something that was only achieved in orchestration scenarios using BPEL. Further work was done to provide a more flexible way of modelling complex conversations. This was achieved by implementing a declarative meta-model for interaction protocols inspired by OWL-P (Desai et al. 2005*b*). This approach makes protocol models highly modular and reusable, but more importantly, details like how to handle the change in the cardinality of the participants, and when to synchronize the otherwise parallel running conversations are now more easy to model. The model inherits much traditional forward planning, but more importantly, using the declarative approach, concepts are easier to relate. In our case, Commitments, Timeouts and Speech Acts are very clearly harmonized to represent these fundamental concepts of interaction protocols. New constructs are easier to add making the meta-model very scalable. The modeling approach was used to model an industrial use case with great success, since it enabled modelling of the complete process combining only two basic interaction protocols. The result was a model that performs as described in the use case. The model can be used for other implementations, in our case, a transformation to Jadex was used to produce BDI agents that can communicate using Web Services to perform the business process.

Chapter 10

Future Work

Once the declarative meta-model for interaction protocols was implemented and produced executable agents that perform these models using Web Services, some open problems were recognized. The next two sections will list some open issues and future work.

10.1 Agent communication grounded on Web Services

10.1.1 Standardization of contents description

Our work was focused on standardizing the communication protocols for agents to be compatible with Web Services. We deliberately decided to avoid any work with respect to message contents, in contrast to some of the Gateway approaches discussed in Section 9.5.

Concerning message contents compatibility, portability and mapping, there are many techniques and standards being developed or used in practice. Web Services and XML standards are mainly concerned with this. There are approaches strongly oriented towards ontology, like RDF, and very technical and structure-oriented specifications, like XML-Schema, and other similar ones, for which mapping and transformation techniques are well developed. Our work has left open; how these techniques are to be integrated into our framework.

10.1.2 Service description and discovery

We only provide a solution for communication level integration and some other aspects of SOA remain open, like service and content description and service discovery. These two aspects are closely related to one another and are the subject of study areas like Semantic Web Services description and match-making (Klusch, Nesbigall & Zinnikus 2008).

Our work, as shown in Figure 1.1 in Section 1.3, is targeted at implementing business processes. The experience gained in projects like ATHENA (Benguria et al. 2007) and SHAPE (Benguria et al. 2009) show the strong interest in implementing business processes following a life cycle that is conceptually in the opposite direction to traditional SOA life-cycles. In SOA, firstly, desirable participants are described and then searched, the discovery phase. Secondly, matching participants are contacted, their interfaces matched and a process is composed as a result. Business processes are first modeled, out of these models interfaces are generated which, afterwards, are implemented by the concrete participants that will perform the process. As we can see, this is a development process where discovery of services plays a minor role. Details that are important to mention in a description of participants are information about their reasoning power or their capabilities. One first approach, as proposed by FIPA, would be to mention the Speech Acts, protocols and ontologies that the participant can process. This aspect will be discussed next.

10.1.3 Heterogeneous Web Service agents

JadeWSMTS clears barriers at the levels of message transport and representation. Agents in a multi-agent system interact with the assumption that other agents can understand their messages and are capable of reasoning about them.

Interaction between parties with different reasoning capabilities tends to reduce the overall communication capacity to levels that can be as low as that of the least capable participant, as it happens with some gateways.

In the case of peer to peer (P2P) the notion of peer implies that all participants share the same reasoning power. In contrast, in some multi-agent systems composed of agents of heterogeneous complexity, the description of agents and annotations in the ACLMessages, about protocols and ontologies the agent can process, let participants know the communicative capabilities of their counterpart.

As mentioned in the previous sub-section, the first step in coping with this issue is to let the participants be aware of what other participants can do. We propose to follow FIPA's proposition of integrating this in the description of agents.

10.1.4 Stateful Web Services

Other open issues are those related to the stateful nature of agents. The concept of anonymity can be a challenge for agent implementations, because agents require other mechanisms to differentiate anonymous parties in a conversation. The traditional stateless nature of services can have stronger repercussions at the time of performing complex conversations since it relies

some times on the concept of Commitments. This normally implies that parties manage different states during the conversation. Some Web Services support this, at least in certain sense, by using sessions or similar concepts. It is relevant to study the relationship between stateful entities, Commitments and complex conversations and also to compare messaging mechanisms like REST and robust messaging like in FIPA or WS-Addressing.

10.1.5 Identification of Services

Finally, it is also important to consider the use of a unique identification for services when the systems that will interact are unknown and could have different identification assigning mechanisms. This issue is very common in peer to peer networks of heterogeneous systems (Josephson, emin Gün Sirer & Schneider 2004) (Fritz & Páris 2004) (Zhang & Zhang 2004). Relevant work has been done in this area using FIPA agents and porting such a solution to this integrated architecture would provide interesting insights for Web Services based peer to peer networks.

10.2 Declarative meta-model of interaction protocols

The present work succeeded in providing a model and tools to produce descriptions of protocols in a declarative way, however, there are, among others, two areas for future study: what other proposition names or constructs are necessary and how to make the graph diagram more intuitive, expressive and usable.

10.2.1 Modeling constructs

Declarative approaches profit from expressiveness, given their descriptive nature. In the present work some basic constructs have been defined to describe situations and conditions in a conversation like: Timeouts, Commitments, Conditional Propositions, Cross-Conversational Constraints, etc.

Some of these concepts have always been used in conjunction with interaction protocols, like Timeouts. Some other came about afterwards and have a generous amount of literature supporting them, like Commitments. Some were defined concretely for the first time to clear existing ambiguities in interaction protocol models and put to work along with other constructs as part of this work, like Cross-Conversational Constraints.

It is still not clear what set of concepts are a minimum requirement in order to describe conversation protocols appropriately. Further study will be needed to evaluate the proposed constructs or to come up with a compelling set of fundamental constructs to be used in a Declarative Protocol meta-model.

10.2.2 Diagram constructs

Similarly to the previous section, where we proposed to evaluate and even extend the set of constructs used to produce our Declarative Protocol meta-model, the graphical elements used in the diagram editor need further study and improvement. The current editor produces diagrams for protocols in the form of directed graphs which, given their flexibility match, the nature of a conversation better in opposition to procedural structures, which are inadequate for this task. Even so, and as already mentioned in (León Soto 2009), there are some structural patterns typical to protocols and workflow modelling that are not easy to recognize in the graph.

Our diagrams show only the flow of the conversations in a very basic manner: in a conversation. There are, for instance, patterns that make the conversation (with the same partner) more recognizable and structured. Situations, like parallel flows inside the same conversation, State Descriptions where conversations split or are joined, are common phenomenons that can be defined using our theoretical meta-model and correspondingly annotated or displayed in a recognizable way in the diagram. Maybe, some ideas for visual representation could be introduced in our diagrams, for instance those of BPMN (see Sections 3.5.3 and 9.5.2), since it was developed with the objective of making intuitive and expressive diagrams.

More graphical elements and adding the logic to the editor to find such situations and representing them accordingly would make the editor more usable and understandable. A content rich model like the one used here can only be of advantage if the user recognizes and has access to all this information. Graphical elements can help dramatically and more research and design should be invested in developing this aspect.

Also, after using the diagram editor, one recognizes tasks that are rather repetitive and could be automatized. Our framework allows such improvements and, at the same time, these improvements are necessary to make our diagrams more usable.

10.2.3 Standardization of Speech Acts and protocols in a library

One of the goals pursued in this thesis and inspired by the goals of the multi-agents community is to count with a set of standardized elements, by combining them, to solve most of the problems.

FIPA attempted that by proposing a Speech Acts and an interaction protocols library. Our meta-model serves this same objective, therefore, the proposal of a core library of Speech Acts and interaction protocols would make much sense, specially since this would be one of the best ways to take advantage of the reuse of code and modularity features of our meta-model.

For this purpose a set of Speech Acts is defined providing them with

enough semantic description in the form of our declarative model to use them flexibly in several scenarios. Using these Speech Acts, fundamental interaction protocols could be defined. As FIPA showed, this is a very hard task and it involves the whole multi-agents community. Our meta-model would help the community to develop these models.

10.2.4 Semantic matchmaking

Our models are well annotated with semantic information. Our tool makes use of this in a trivial manner, by simply doing a lexical matching of proposition names. However, in comparison to how it is done in Semantic Web Services, more sophisticated matchmaking tools and algorithms could be used, to improve the performance, accuracy and sophistication of our model editors. How our framework would fit in such a scenario is left open for further research.

10.2.5 Reasoning about interaction protocols and actions

Our meta-model provides information initially targeted at the improvement of the design tools and the management of models. This kind of models increase interest in the multi-agent systems community for implementing agents that can reason about the annotations available in the models.

This would bring our models closer to dialog-games. Our models count with key information, the semantics of actions and protocols. We used it to support the design process, but at run-time, it would also be of benefit for agents capable of reasoning about what protocols, actions or paths in a conversation to take. The implementation of this kind of agents is a comprehensive research area and we believe our meta-model would make a relevant contribution to this area.

10.3 Summary

Our work brings forward new questions and areas where further work is proposed. We left the area of message contents standardization or mapping in our Web Services and agents integration open. Because of alignment with our objectives the description and discovery of participants was ignored since they do not play a role in the development of business process models approached in this thesis. Also some further work would be useful concerning the heterogeneous nature or participants in Web Services, their statefulness and the effects and obstacle of identification mechanisms.

We proposed interaction protocols constructs that are formally related to the whole meta-model. Some new concepts were provided, like Cross-Conversational Constraints, but other constructs and how they are related to the meta-model raises interest. The ultimate goal of having a canonical

library of speech acts and interaction protocols is still open. Our meta-model is suitable for more intense use of its semantics: semantic match making of protocols and the implementation of agents capable of reasoning about how to interact, based on the descriptions present in the model, are proposed as future steps to take.

Chapter 11

Conclusions

In this thesis, the goal of using multi-agent communication techniques for the realization of Business-Processes has been achieved by approaching the problem at two levels:

- At the lower level; integrating Web Services and multi-agent systems communication specifications
- At the higher level; modeling of Business-Process using our declarative meta-model for interaction protocols.

Using MDA-techniques, implementations that make use of our achievements in the lower levels are automatically generated using the models produced by the tools provided in the upper level as input.

As a result, we have a set of tools that aid in the design of Business-Processes by using the interaction protocols based approach of multi-agents systems and automatically generating implementations of these complex conversations using Web Services standards.

The experience of the projects; (Benguria et al. 2007),(Benguria et al. 2009) and (Hahn, Madrigal Mora & Fischer 2009), guided us to also use a design life-cycle different from the traditional SOA approach. Our implementation is not the result of discovery of services, but instead we start with a design of a desired process, we generate the interfaces for the participants and produce implementations of them in order to perform the modeled process.

11.1 Choreography

The models and implementations always respect the independence of the participants. Models are globally seen choreographies and do not include any concept of a ruling or guiding entity. Implementations are created based on the interaction models in such a way that they fit together at

run-time and perform as specified in the design. The only element keeping the business-process working correctly is the model they are based on, giving our interaction protocol models the role of *contract* between participants that defines how these participants agree to act together.

11.2 A declarative approach

Our intention was to have models that were modular, reusable and with features that suit the nature of interactions between autonomous participants: Commitments, Timeouts, Cross-Conversational Constraints (see 5.4.4), etc.

A declarative approach gives more freedom to express the flexibility of a conversation instead of forcing it into programming constructs as usually happens in traditional process modeling frameworks.

11.2.1 Modularity and reuse

Our modeling of Business Processes is inspired by the way the multi-agents systems community, and more specifically FIPA, intended to solve this problem.. This was done by providing a set of Speech Acts and interaction protocols that by combination can be used to solve most situations. We overcome some of the difficulties of FIPA specifications by defining Speech Acts and interaction protocols using the same semantics. We profit from coherence throughout the meta-model, in opposition to FIPA, where a dialog-game approach was used for defining Speech Acts and a global perspective for the interaction protocols.

11.2.2 Consolidation of contributions

Before implementing the meta-model, we produced a comprehensive, formal and detailed semantic definition of our concepts (León Soto 2009). The set of definitions were necessary to consolidate contributions of the multi-agents systems community. Many ideas common in the community are included, like Timeouts and Commitments, and new definitions are included like Cross-Conversational Constraints. Some of these concepts were still defined ambiguously in literature and in our definition we disambiguated them and established clear relations among them that define how all the concepts work together. To achieve this, many techniques well known in artificial intelligence, like forward planing, were used.

11.3 JadeWSMTS

We provide a tool called JadeWSMTS that integrates Agents and Web Services. It does not change the nature of agents and Web Services, but provides tools and techniques that let them work together. Thanks to a mapping of

Web Services and FIPA messaging properties, agent messages are represented using Web Services standards. In this way, agents and Web Services can interact with each other transparently. Still agents might need to adapt their internal operations in order to interact with Web Services. Our tool helps in making stateless or synchronous Web Services, and stateful and asynchronous agents compatible. However, it cannot overcome the difference in reasoning power that might arise between participants in a scenario with conventional Web Services.

11.4 Automatic generation of flexible and scalable implementations

Out of our models Jadex BDI agents are created automatically. The agents can communicate using JadeWSMTS. The conversion takes advantage of the declarative aspect of the models, and produces a context based conversation control implementation. BDI agents make the situation oriented approach easier to implement than using traditional procedural techniques. At the same time, a goal oriented pattern of plans is used to implement the roles. This makes the implementation of agents better in terms of flexibility and scalability. The normal approach, even in PIM4Agents (Hahn, Madrigal Mora & Fischer 2009), is to produce a single segment of code for the complete behaviour of the role. Our code generation can produce goal oriented implementations of the BDI agents, where each turn in the conversation is implemented using a small set of plans, thanks to the declarative nature of the meta-model. The implementation of a role in BDI agents is therefore goal oriented and flexible instead of rigid sequences of code, like those used in traditional transformations.

Bibliography

(2003).

Agentcities (2002), 'Agentcities', http://www.agentcities.org.

Agentcities Web Services Working Group (2002), 'Integrating Web Services into Agentcities'. http://www.agentcities.org/Activities/WG/WebServices/.

Amgoud, L., Maudet, N. & Parsons, S. (2000), Modeling dialogues using argumentation, *in* 'Proceedings of the Fourth International Conference on MultiAgent Systems'.

Andrews, T. & et. al., F. C. (2003), Business Process Execution Language for Web Services, Technical report, IBM.
URL: *ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf*

Ardissono, L., Goy, A. & Petrone, G. (2003), Enabling conversations with web services, *in* 'AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems', ACM Press, New York, NY, USA, pp. 819–826.

AXIS2 (2006), 'Axis2 SOAP Stack implementation', http://ws.apache.org/axis2/.

Barros, A., Dumas, M. & Oaks, P. (2005), 'A Critical Overview of the Web Services Choreography Description Language'. www.btrends.com.

Barry, D. K. (2003), *Web Services and Service-Oriented Architectures*, Morgan Kaufmann.

Bauer, B., Müller, J. P. & Odell, J. (2000), Agent UML: A Formalism for Specifying Multiagent Software Systems, *in* 'ICSE 2000 Workshop on Agent-Oriented Software'.

Benguria, G., Berre, A. J., Elvesæter, B., Hahn, C., Jacobi, S., Landre, E., Sadovykh, A. & Stollberg, M. (2009), 'SHAPE Project Whitepaper'.

Benguria, G., Larrucea, X., Elvester, B., Neple, T., Beardsmore, A. & Friess, M. (2007), A platform independent model for service oriented architectures, *in* G. Doumeingts, J. Mller, G. Morel & B. Vallespir, eds, 'Enterprise Interoperability', Springer London, pp. 23–32.

Braubach, L. & Pokahr, A. (2007), Goal-oriented interaction protocols, *in* 'Fifth German conference on Multi-Agent System Technologies (MATES-2007)'.

Curbera, F., Khalaf, R., Mukhi, N., Tai, S. & Weerawarana, S. (2003), 'The next step in web services', *Communications of the ACM* **46**(10), 29–34.

Dale, J. & Lyell, M. (2003), Services work plan, Technical report, Foundation for Intelligent Physical Agents (FIPA).

Damodaran, S. (2004), B2B integration over the Internet with XML: RosettaNet successes and challenges, *in* 'WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters', ACM, pp. 188–195.

Decker, G., Kopp, O., Leymann, F., Pfitzner, K. & Weske, M. (2008), Modeling Service Choreographies using BPMN and BPEL4Chor, *in* '20th International Conference on Advanced Information Systems Engineering (CAiSE08)', Springer, pp. 79–93.

Decker, G., Kopp, O., Leymann, F. & Weske, M. (2007), BPEL4Chor: Extending BPEL for Modeling Choreographies, *in* 'ICWS 2007', IEEE Computer Society, pp. 296–303.

DeRemer, F. & Kron, H. (1975), Programming-in-the large versus programming-in-the-small, *in* 'Proceedings of the international conference on Reliable software', ACM, pp. 114–121.

Desai, N., Chopra, A. K. & Singh, M. P. (2009), *AMOEBA: A Methodology for Modeling and Evolution of Cross-Organizational Business Processes*, Vol. 19, ACM.

Desai, N., Mallya, A. U., Chopra, A. K. & Singh, M. P. (2005*a*), 'Interaction Protocols as Design Abstractions for Business Processes', *Transactions on Software Engineering* **31**(12), 1015 – 1027.

Desai, N., Mallya, A. U., Chopra, A. K. & Singh, M. P. (2005*b*), OWL-P: A Methodology for Business Process Modeling and Enactment, *in* 'Workshop on Agent Oriented Information Systems', pp. 50–57.

Desai, N. & Singh, M. P. (2007), A Modular Action Description Language for Protocol Composition, *in* 'AAAI', pp. 962–967.

DFKI (2007), 'FIPA SL representation in XML', http://www.dfki.de/∼estebanl/JadeWSMTS/FIPA-XML-SL.xsd.

Ehrler, L. & Cranefield, S. (2004), Executing Agent UML Diagrams, *in* 'AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems', IEEE Computer Society, pp. 906–913.

Eijk, R. M. V., Boer, F. S. D., Hoek, W. V. D. & Meyer, J.-J. C. (2003), 'A verification framework for agent communication', *Autonomous Agents and Multi-Agent Systems* **6**(2), 185–219.

FIPA (2002), 'Foundation for Intelligent Physical Agents'. http://www.fipa.org.

FIPA (2002*a*), 'FIPA Abstract Architecture Specification'.

FIPA (2002*b*), 'FIPA Agent Management Specification'.

FIPA (2002*c*), FIPA Agent Message Transport Envelope Representation in XML Specification, Technical report, Foundation For Intelligent Physical Agents (FIPA), http://www.fipa.org/specs/fipa00071/.

FIPA (2002*d*), FIPA Agent Message Transport Services Specification, Technical report, (FIPA) Foundation for Intelligent Physical Agents, http://www.fipa.org/specs/fipa00067.

FIPA (2002*e*), 'FIPA Communicative Act Library Specification'. http://www.fipa.org/specs/fipa00037.

FIPA (2002*f*), FIPA Contract Net Interaction Protocol Specification, Technical Report SC00029H, FIPA.
URL: *http://www.fipa.org/specs/fipa00029*

FIPA (2002*g*), FIPA Interaction Protocols Specification, Technical report, Foundation For Intelligent Physical Agents (FIPA), http://www.fipa.org/repository/ips.php3.

FIPA (2002*h*), 'FIPA SL Content Language Specification'.

FIPA (2004), 'FIPA Agent Management Specification'. http://www.fipa.org/specs/fipa00023.

Fischer, K., Kuhn, N., Müller, H.-J. & Müller, J. P. (1995), 'Modelling the transportation domain', *Computational Economics* **8**, 81–93.

Forrester, J. (1969), *Industrial Dynamics*, 6. print edn, M.I.T. Press, Cambridge, Mass.

Fritz, A. & Páris, J.-F. (2004), Maille Authentication, A Novel Protocol for Distributed Authentication, *in* 'Security and Protection in Information Processing'.

Gibbins, N., Harris, S. & Shadbolt, N. (2003), Agent-based semantic web services, *in* 'Proceedings of the twelfth international conference on World Wide Web', ACM Press, pp. 710–717.

Greenwood, D. & Calisti, M. (2004), Engineering web service - agent integration, *in* 'Systems, Cybernetics and Man Conference', Whitestein, IEEE.

Greenwood, D., Lyell, M., Mallya, A. & Suguri, H. (2007), The IEEE FIPA Approach to Integrating Software Agents and Web Services, *in* 'Sixth International Conference on Autonomous Agents and Multiagent Systems, Industrial Track'.

Greenwood, D., Nagy, J. & Calisti, M. (2005), 'Semantic Enhancement of a Web Service Integration Gateway', *Workshop on Services-Oriented Computing and Agent-Based Engineering at AAMAS 05* .

Hahn, C. & Fischer, K. (2008), The formal semantics of the domain specific modeling language for multiagent systems, *in* 'Agent-Oriented Software Engineering IX, 9th International Workshop, AOSE', Vol. 5386 of *Lecture Notes in Computer Science*, pp. 145–158.

Hahn, C., Madrigal Mora, C. & Fischer, K. (2009), 'A platform-independent metamodel for multiagent systems', *Autonomous Agents and Multi-Agent Systems* **18**(2), 239–266.

Hahn, C., Zinnikus, I., Warwas, S. & Fischer, K. (2009), From agent interaction protocols to executable code: a model-driven approach, *in* 'AAMAS (2)', pp. 1199–1200.

Haugen, O. & Runde, R. K. (2009), *Agent-Based Technologies and Applications for Enterprise Interoperability*, Vol. 25 of *LNBIP*, Springer, chapter Enhancing UML to Formalize the FIPA Agent Interaction Protocol, pp. 154–173.

Havey, M. (2005), *Essential Business Process Modeling*, O'Reilly.

Hohpe, G. & Woolf, B. (2003), *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Professional.

IETF (2004), 'Extensible Messaging and Presence Protocol (XMPP) RFC 3920'. http://xmpp.org/.

Jacobi, S., León Soto, E., Madrigal Mora, C. & Fischer, K. (2005), AgentSteel: an agent-based online system for the planning and observation of steel production, *in* 'AAMAS Industrial Applications', Utrecht, pp. 114–119.

Jacobi, S., León Soto, E., Madrigal Mora, C. & Fischer, K. (2007), MasDISPO: a multiagent decision support system for steel production and control, *in* 'Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence', AAAI, Vancouver, pp. 1707–1714.

JADE (2001), 'Java Agent Development Framework', On line. http://jade.tilab.com.

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

Jilovec, N. (2004), *EDI, UCCnet and RFID: synchronizing the supply chain* , Penton Technology Media.

Josephson, W. K., emin Gün Sirer & Schneider, F. B. (2004), Peer-to-Peer Authentication with a Distributed Single Sign-On Service, *in* 'Peer-to-Peer Systems III', Vol. 2872 of *LNAI*.

Kantor, M. & Burrows, J. H. (1993), Electronic data interchange (edi), Technical report, National Institute of Standards and Technology. http://www.itl.nist.gov/fipspubs/fip161-2.htm.

Klusch, M., Nesbigall, S. & Zinnikus, I. (2008), MDSM: Model-Driven Semantic Web Service Matchmaking for Collaborative Business Processes. P, *in* 'Proceedings of IEEE/ACM International Conference on Web Intelligence (WI)', Vol. 1, pp. 612 – 618.

Kramler, G., Kapsammer, E., Retschitzegger, W. & Kappel, G. (2006), *Towards Using UML 2 for Modelling Web Service Collaboration Protocols*, Springer London, chapter 21, pp. 227–238.

León Soto, E. (2006), FIPA Agents Messaging grounded on Web Services, *in* 'Grid Services Engineering and Management (GSEM)', Vol. P-88 of *LNI*.

León Soto, E. (2007), Agent Communication Using Web Services, a new FIPA Message Transport Service for Jade, *in* P. Petta, J. P. Müller, M. Klusch & M. Georgeff, eds, 'Multiagent System Technologies', Vol. 4687 of *LNAI*, Springer, pp. 73–84.

León Soto, E. (2009), Modelling Interaction Protocols as Modular and Reusable 1st Class Objects, *in* K. Fischer, J. P. Müller, J. J. Odell & A. J. Berre, eds, 'Agent-Based Technologies and Applications for

Enterprise Interoperability', Vol. 25 of *LNBIP*, Springer, chapter 10, pp. 174–219.

León Soto, E. (2012), A Model-Driven Approach for Executing Modular Interaction Protocols Using BDI-Agents, *in* K. Fischer, J. P. Mller, R. Levy, W. Aalst, J. Mylopoulos, M. Rosemann, M. J. Shaw & C. Szyperski, eds, 'Agent-Based Technologies and Applications for Enterprise Interoperability', Vol. 98 of *Lecture Notes in Business Information Processing*, Springer, pp. 10–34.

Mallya, A. U. & Huhns, M. N. (2003), 'Commitments Among Agents', *IEEE Internet Computing* **7**(4), 91–94.

Mallya, A. U. & Singh, M. P. (2004), A Semantic Approach for Designing E-Business Protocols, *in* 'Third International Conference on Web Services', IEEE Computer Society, pp. 742–745.

Mallya, A. U. & Singh, M. P. (2005), A Semantic Approach for Designing Commitment Protocols, *in* R. V. Eijk, ed., 'Developments in Agent Communication', Vol. 3396 of *LNAI*, Springer, pp. 37–51.

Mallya, A. U. & Singh, M. P. (2006*a*), Specifying and resolving preferences among agent interaction patterns, *in* 'AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems', ACM, pp. 1361–1368.

Mallya, A. U. & Singh, M. P. (2006*b*), Specifying and resolving preferences among agent interaction patterns, *in* 'In Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems', ACM Press, pp. 1361–1368.

McBurney, P. & Parsons, S. (2002), 'Games that agents play: A formal framework for dialogues between autonomous agents', *J. of Logic, Lang. and Inf.* **11**(3), 315–334.

McBurney, P. & Parsons, S. (2003), 'Dialogue game protocols', *Communications in Multiagent Systems* **2650**, 269–283.

Miller, T. & McBurney, P. (2007), Using constraints and process algebra for specification of first-class agent interaction protocols, *in* 'Engineering Societies in the Agents World VII', Vol. 4457 of *LNAI*, p. 245264.

Miller, T. & Mcginnis, J. (2008), Amongst first-class protocols, *in* 'Engineering Societies in the Agents World VIII: 8th International Workshop, ESAW 2007, Athens, Greece, October 22-24, 2007, Revised Selected Papers', Springer-Verlag, pp. 208–223.

Nguyen, X. T. & Kowalczyk, R. (2005), 'WS2JADE: Integrating Web Services with Jade Agents', *Workshop on Services-Oriented Computing and Agent-Based Engineering at AAMAS 05* .

OASIS (n.d.), 'Universal Description, Discovery and Integration (UDDI)'. http://uddi.xml.org/.

Odell, J., van Dyke Parunak, H. & Bauer, B. (2000), Representing Agent Interaction Protocols in UML, *in* 'International Workshop on Agent-Oriented Software Engineering'.

OMG (2008), 'Querry view transformation', http://www.omg.org/spec/QVT/1.0/PDF/.
URL: *http://www.omg.org/spec/QVT/1.0/PDF/*

OMG (2009), Business Process Modeling Notation (BPMN), Technical report, Object Management Group. http://www.omg.org/spec/BPMN.

Palanca, J., Escrivá, M., Aranda, G., García-Fornes, A., Julian, V. & Botti, V. (2006), Adding New Communication Services to the FIPA Message Transport System, *in* 'Multiagent System Technologies (MATES)', Vol. 4196 of *LNCS*, Springer.

Pokahr, A., Braubach, L. & Lamersdorf, W. (2005*a*), Jadex: A bdi reasoning engine, *in* J. D. R. Bordini, M. Dastani & A. E. F. Seghrouchni, eds, 'Multi-Agent Programming', Springer Science+Business Media Inc., USA, pp. 149–174. Book chapter.

Pokahr, A., Braubach, L. & Lamersdorf, W. (2005*b*), Jadex: A BDI reasoning engine, *in* M. D. R. Bordini & A. E. F. Seghrouchni, eds, 'Multi-Agent Programming', Springer Science+Business Media Inc., USA, pp. 149–174. Book chapter.

Raber, D. (2009), A Model-Driven Approach for the Integration of Multiagent Systems and Service-Oriented Architectures in the Steel Industry, Master's thesis, University of Saarland.

RossettaNet (2002). http://www.rosettanet.org.

Saarstahl AG (n.d.), 'Saarstahl AG', On line. http://www.saarstahl.com.

Sadek, M. (1991), 'Dialogue Acts are Rational Plans', *ESCA/ETRW Workshop on the Structure of Multimodal Dialogue* .

Service oriented architecture Modeling Language (2008), Revised OMG submission.

Shafiq, M. O., Ali, A., Ahmad, H. F. & Suguri, H. (2005), Agentweb gateway - a middleware for dynamic integration of multi agent system and web services framework, *in* 'WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise', IEEE Computer Society, pp. 267–270.

Shehory, O. (2003), Robustness Challenges in Peer-to-Peer Agent Systems, *in* 'Agents and Peer-toPeer Computing'.

Shin, H.-J. & Lee, S.-G. (2004), Architecture environtments for e-business agent based on security, *in* 'Computatoinal Science and its Applications', Springer.

Singh, M. P. (1999), 'An Ontology for Commitments in Multiagent Systems', *Artificial Intelligence and Law* **7**(1), 97–113.

Singh, M. P., Chopra, A. K., Desai, N. V. & Mallya, A. U. (2004), Protocols for Processes: Programming in the Large for Open Systems, *in* 'OOPSLA Companion', Vol. 39 of *ACM SIGPLAN Notices*, ACM, pp. 73–83.

Singh, M. P. & Huns, M. N. (2005), *Service-Oriented Computing Semantics, Processes and Agents*, WILEY.

Smith, H. & Fingar, P. (2002), *Business Process Management, the third wave*, Megan-Kiffer Press.

Smith, R. G. (1979), The Contract Net Protocol: High-Level Communication And Control In A Distributed Problem Solver, *in* 'Proceedings of the First International Conference On Distributed Computing Systems', pp. 185–192.

Sonntag, M. (2006), 'Agents as Web Service providers: Single agents or MAS?', *Applied Artificial Intelligence* **20**, 203–227.

van den Heuvel, W.-J. & Maamar, Z. (2003), Moving toward a framework to compose intelligent web services, Vol. 46, pp. 103–109.

van Riemsdijk, M. B., Dastani, M., Meyer, J.-J. C. & de Boer, F. S. (2006), Goal-oriented modularity in agent programming, *in* 'AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems', ACM, pp. 1271–1278.

W3C (2002), 'Web Services Description Language'. http://www.w3.org/2002/ws/desc/.

W3C (2003), 'SOAP Simple Object Access Protocol'. http://www.w3.org/TR/soap/.

W3C (2005), 'Web Services Choreography Description Language'. http://www.w3.org/TR/ws-cdl-10/.

W3C (2006*a*), 'Web services addressing', http://www.w3.org/2002/ws/addr/.

W3C (2006*b*), 'Web Services Addressing 1.0-Core', http://www.w3.org/TR/ws-addr-core.

W3C (2006*c*), 'Web Services Addressing 1.0-SOAP Binding', http://www.w3.org/TR/ws-addr-soap.

W3C (n.d.), 'World Wide Web Consortium'. http://www.w3.org.

Warwas, S. & Hahn, C. (2009), The dsml4mas development environment, *in* C. Sierra, C. Castelfranchi, K. S. Decker & J. S. Sichman, eds, 'AAMAS (2)', IFAAMAS, pp. 1379–1380.

Warwas, S., Hahn, C. & Fischer, K. (2009), A visual development environment for jade, *in* 'AAMAS (2)', pp. 1349–1350.

Weerawarana, S., Curbera, F., Leymann, F., Storey, T. & f. Ferguson, D. (2005), *Web Services Platform Architecture*, Prentice Hall.

Winikoff, M. (2005), Jack intelligent agents: An industrial strength platform, *in* J. D. R. Bordini, M. Dastani & A. E. F. Seghrouchni, eds, 'Multi-Agent Programming', Springer Science+Business Media Inc., USA, pp. 175–196. Book chapter.

Yolum, P. & Singh, M. P. (2002), Commitment machines, *in* 'ATAL '01: Revised Papers from the 8th International Workshop on Intelligent Agents VIII', Springer-Verlag, pp. 235–247.

Zhang, Y. & Zhang, D. (2004), Authentication and Access Control in P2P Network, *in* 'Grid and Cooperative Computing'.