

# Developing a Negotiation Protocol for ORS

*Andriana Evgenia Gkaniatsou*



Master of Science  
Artificial Intelligence  
School of Informatics  
University of Edinburgh  
2011

# Abstract

In this thesis we present a negotiation protocol in the context of the Ontology Repair System (ORS). Our aim is to enable agents to negotiate in real-time over potential ontology repairs. Based on this goal, we devise, implement, and evaluate an agent negotiation model. The model's goal is to orchestrate the negotiation, thus allowing the agents to reach an agreement regarding the allocation of the repair. In doing so we achieve a rich and efficient interaction. At the same time, the performance of the original system is not penalised in any way: the performance of the resulting system is always at least as good as that of the original ORS as far as plan execution is concerned.

# **Acknowledgements**

I owe my deepest gratitude to my supervisor, Fiona McNeill, for her insight, support and encouragement throughout this project. I would also like to thank my second supervisor Liwei Deng for actively participating in our meetings and his feedback. I am deeply grateful to Pagoni for the emotional support and for bearing my moaning. Last but not least, I am deeply grateful to my parents for boosting me morally.

# Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

*(Andriana Evgenia Gkaniatsou)*

To my parents and Theos

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background</b>	<b>5</b>
2.1	Multi-agent systems . . . . .	5
2.1.1	Agent Communication . . . . .	5
2.1.2	Agent negotiation . . . . .	7
2.2	Ontologies . . . . .	9
2.2.1	Ontology evolution . . . . .	10
2.3	Negotiation Ontology . . . . .	11
2.4	Automated planning . . . . .	12
2.5	Ontology Repair System . . . . .	13
<b>3</b>	<b>Negotiation Protocol</b>	<b>17</b>
3.1	Introduction . . . . .	17
3.2	Negotiation Domain . . . . .	18
3.2.1	Negotiation Parties . . . . .	19
3.2.2	Negotiation Object . . . . .	19
3.2.3	Cost Function and Utility . . . . .	21
3.2.4	The Negotiation Offer and the Negotiation Strategy . . . . .	25
3.2.5	Negotiation Agreement . . . . .	25
3.2.6	Expert Agent . . . . .	27
3.3	Negotiation Process . . . . .	28
3.3.1	The negotiation messages and the corresponding negotiation phases . . . . .	33
3.3.2	Negotiation rules and obligations . . . . .	34
3.4	Summary . . . . .	38

<b>4</b>	<b>Negotiation Ontology</b>	<b>40</b>
4.1	Introduction . . . . .	40
4.2	Basic Concepts . . . . .	41
4.3	Concept Relations and Instances . . . . .	44
4.3.1	Negotiation Message . . . . .	44
4.3.2	Negotiation Party . . . . .	45
4.3.3	Negotiation Object . . . . .	47
4.3.4	Negotiation Phase . . . . .	47
4.3.5	Rules . . . . .	49
4.3.6	Summary . . . . .	49
<b>5</b>	<b>Implementation</b>	<b>51</b>
5.1	Overview of the system . . . . .	51
5.2	The Negotiation Service . . . . .	52
5.2.1	Consulting the expert agent . . . . .	56
5.2.2	Negotiation Ontologies . . . . .	60
5.2.3	Planning System . . . . .	63
5.3	Communication System . . . . .	68
5.3.1	Processing the negotiation ontology . . . . .	70
5.4	Forming the utility . . . . .	72
5.5	Summary . . . . .	73
<b>6</b>	<b>Evaluation</b>	<b>74</b>
6.1	Aims of the Evaluation . . . . .	74
6.2	Negotiation Process evaluation . . . . .	75
6.2.1	The Economy scenario . . . . .	76
6.2.2	Water Management scenario . . . . .	83
6.2.3	Timings . . . . .	86
6.2.4	Summary . . . . .	88
6.3	Negotiation Protocol Evaluation . . . . .	89
6.3.1	Negotiation protocol implementation evaluation . . . . .	93
6.3.2	Summary . . . . .	94
<b>7</b>	<b>Related work</b>	<b>96</b>
<b>8</b>	<b>Conclusions and future work</b>	<b>99</b>

<b>A Economy Scenario</b>	<b>101</b>
<b>B Protocol evaluation</b>	<b>111</b>
<b>C Console outputs</b>	<b>113</b>
<b>Bibliography</b>	<b>115</b>

# Chapter 1

## Introduction

Agent communication is the essential means the agents use to exchange information. Effective communication is a result of the agreement between agents about the terminology that is going to be used. This ensures messages are mutually understood. Ontologies provide the necessary terminology for the specification of an agent's domain of interest and knowledge of its environment. So, communication between agents is possible when the parts of the ontologies that are utilised for the communication do not differ. To aggravate things, in open multi-agent systems, the agents do not follow a common design principle. This implies that the vocabulary and the conceptualisation each utilised ontology represents can vary widely. Ontologies differ depending on the domain the agent is modeled for and the purposes of the agent, as well as the ontology engineering.

Ontologies are the logical theories that are used to describe the knowledge of agents and the perception of their environment. When two or more agents interact, any possible ontological mismatch can lead to communication failure. In open multi-agent systems, where ontologies are continuously changing and the agents come from different sources, ontological mismatches can be considered as an inevitable feature [Finkelstein et al., 1993]. These mismatches are ontological dissimilarities caused by the precision of the specifications, the used vocabulary, and the version of the ontology. A lot of research has been done on this topic, and different approaches have been proposed for mismatch diagnosis and ontology refinement. One of these approaches [Bundy and McNeill, 2006] deals with dynamic repair of ontologies in real time whenever a mismatch occurs. An output of this approach is the Ontology Repair System.

The Ontology Repair System (ORS) investigates the mismatch problem in a plan-

ning context. Its intent is to solve the problem by dynamically repairing the ontologies of the planning agent (henceforth PA). In this way it enables the PA to communicate with the service providing agents (henceforth SPAs) despite any ontological mismatches. When a communication problem among any two agents occurs, ORS will first diagnose the particular ontological mismatch, and will then repair the ontology of the PA. In this way, the two ontologies are more aligned in this crucial area. The communication process can continue smoothly, until a possible next mismatch.

Until [Bomersbach, 2011], when the first basic approach was made, one of the basic assumptions made within ORS is that the PA always considers the content of an SPA ontology as valid; thus, it is always willing to make the appropriate changes to align its ontology with the other agent's. This implies that SPAs always have more objective ontologies compared to the PA, and that SPAs are neither concerned with altering their own, nor are they aware of the ontological changes the PA is subjected to. This approach corresponds mainly to the characteristics of closed systems. Therefore, it does not reflect the characteristics and needs of open multi-agent systems where data flow is not customised. As a consequence we cannot be certain about the validity of the ontology of the SPA, or if this ontology is up to date.

The ontological repair process in open multi-agent systems should be a matter for both PAs and SPAs for several reasons. Firstly, the data flow within these systems is not fixed since agents are influenced by different environmental factors. Because of that, cases of outdated and different ontologies are very common. Secondly, cases of ontology protection should be considered: ontology protection aims to represent the cost or/and the risk an ontological alteration might bring to the system. For this reason it is not sensible to assume that the PA can always perform the repair. So in the presence of protection the SPA should also consider performing it. We also consider the cases where both the PA and the SPA benefit through their interaction. In these cases the interaction between the PA and the SPA might be characterised as a trading one. For instance, when the PA asks the SPA to perform the action *book-tickets*, the SPA will be financially rewarded for this action. So, for all these reasons, both the PA and the SPA should consider repairing their ontologies in order to achieve a better communication and succeed in their goals.

The aim of this project is repair-sharing through negotiation; we enable the possibility of both agents considering repairing their ontologies. Our hypothesis is:

It is possible to introduce a more sophisticated interaction between the agents than in the original ORS, by enabling the agents to negotiate over

repair allocation. Throughout all this we guarantee that plan execution proceeds at least as well as in the original ORS.

We introduce to the agents a negotiation protocol specifically designed for the purposes of the domain ORS deals with. In this way we provide the agents with the means to negotiate over the repair suggested by ORS and to come to a mutual agreement about which agent will perform it. By doing so, both agents use ORS as a plug-in. The negotiation protocol specifies the set of legal offers and possible agreements. In addition, it provides the agents with the essential vocabulary and the rules for a successful negotiation.

The negotiation outcome -which agent will perform the repair- is always the result of mutual agreement. The strategy that each agent follows depends entirely on a utility-based model: each agent forms its utility for the negotiated repair, and proceeds to negotiate according to this utility. In this way the agents express their preferences. Moreover we have introduced to the system the notion of the social welfare that each agreement brings to the system: the sum of the agents' utilities for each agreement. The set of possible agreements is specified in a way that maximises the social welfare. In this way we ensure that the agreement that is reached is *pareto optimal*: there is no other possible agreement which benefits more one agent without making the other agent worse off.

We also consider cases of ontology protection. Ontology protection aims to ensure that no alteration takes place in specific ontological parts. The level of protection is proportional to the cost associated with changing that specific ontological part. As an example, that cost might be financial or computational. The current implementation associates the utility of a repair to the level of protection; however the protocol is flexible enough to allow other types of utility definition.

Furthermore, we introduce the idea of an expert agent to the system. We define an expert agent an agent with expertise on the domain of a particular mismatch. For instance, let us consider an ontological mismatch between agents  $A_i$  and  $A_j$ . The agent  $A_i$  has in its ontology the representation *money(amount)* while the agent  $A_j$  has the representation *currency(amount)*. So the mismatch is caused between the (*money, currency*). In this case, an expert agent would be a specialised agent on financial issues, such as a *Bank representative*. The negotiators are enabled to ask the expert agent for advice about the representation it considers as correct. We do not force the agents to ask for advice nor to follow this advice; this decision is entirely up to the agents.

While the approach to common protocol implementation is to hard-code the agents,

we followed an entirely different approach. We chose not to hard-code the agents but instead to create a globally available ontology that contains the protocol. In this way we do not force the agents to follow a specific protocol while we made the whole process extensible. In theory, an agent can choose among different protocols or suggest its own by simply providing an ontological representation for this protocol. In addition this approach allows agents hard-coded with different protocols to negotiate successfully. We have proved that agents already hard-coded with another protocol were able to negotiate using ours.

To summarise, we proved that negotiation in the scope of ORS is possible, while the performance of the system is at least as good as that of the original ORS. In addition, we have shown that negotiation extends the abilities of the system to complete plan execution successfully: we have illustrated cases in which negotiation enables the PA to continue its plan execution, while in the original ORS failed. Moreover, we have enabled the agents to base their decisions, concerning the repair, on better information i.e. the advice of the expert agent. In addition, the agreement that is reached brings the least overall cost since it is based on the social welfare.

In what follows we will explain in detail how this project proceeded at both the theoretical and the implementation levels. In Chapter 2 we present the essential background so the reader can understand the domain and the problems this project deals with. In Chapter 3 we present the negotiation protocol we defined for this project. In Chapter 4 the negotiation ontology is described in detail. The implementation we followed and the problems that we faced during the implementation are described in Chapter 5. Throughout this project we tested our hypothesis and we also compared the performance of our extension to the previous versions of ORS. The evaluation process and the results are provided in Chapter 6. In Chapter 7 we discuss about the related work that has been done and compare it to ours. Finally, in Chapter 7 we present our conclusions and discuss directions for future work that can use this project as starting point.

# Chapter 2

## Background

### 2.1 Multi-agent systems

The term agent is used to describe any individual process that is situated within an environment, senses this environment and performs actions. A further characteristic of such process is intelligence, which refers to the ability of reasoning about the environment and the performed actions. Moreover an agent can be autonomous which is the ability to perform actions without human guidance. There also exist other characteristics that aim to imitate human behaviour such as mobility, sociability and negotiation. The distributed systems that are formed by sets of such processes are called multi-agent systems. In these systems the agents can have either individual or common goals which they aim to achieve through communication.

#### 2.1.1 Agent Communication

Communication is considered to be the interaction between two or more processes (agents) in which messages are exchanged. According to the definition of communication in [Ferber, 1999], at the end of the process at least one of the participants will be affected. Process communication has been given great importance, especially in distributed systems, and many formalisms have been developed to describe the properties and the issues that arise between the communicating systems, [Hoare, 1978], [Milner, 1989]. Agents communicate in order to achieve their predefined goals by exchanging information.

Speech act theory, has greatly influenced agent communication. Based on this theory, communication is conceived as a set of actions which are performed by the

agents as an outcome of their intentions. Every speech act that occurs (an exchanged message) changes the state of the world in the same way physical actions do. A set of performative messages have been identified for the needs of the communication act : *request*, *inform* and *promise* [Austin, 1962]. These correspond to different situations that a speech act occurs. In addition, Austin has defined the preconditions for the successful completion of the performatives as:

- The existence of a conventional procedure under which the circumstances and the interacting parties should be identified.
- The procedure must be executed correctly.
- The act has to be sincere.

In addition [Searle, 1969] has classified the possible types of speech acts into

- Representatives, which include all the speech acts that commit the speaker to the truth i.e. inform.
- Directives, which intend to make the receiver do something i.e. request
- Commissives, which entail that the speaker is committed to perform an action i.e. promise.
- Declarations which correspond to the speech acts that change the state of affairs e.g. declare peace.

In addition [Searle, 1969] has also identified the preconditions of these speech acts. Among them are the preparatory conditions which state the world preconditions in order for the speech act to be applicable and the I/O conditions of a speech act.

The information that is exchanged during agent communication is encoded by the senders and decoded by the receivers so that it can be properly understood. Many approaches have been proposed for agent communication, including ones that emphasise the knowledge that is contained in the exchanged queries, for example the KQML protocol [Finin et al., 1994]. Another approach is the one that focuses on the syntax of the exchanged queries such as KIF [Genesereth, 1991]. The difference between these two languages is that the former mostly deals with the syntactically aspects of the messages while the latter commits contents of messages.

However, all the approaches focus on a main principle: interchange of information that each agent posses. Thus, to achieve an efficient communication the agents have to

exchange messages using a shared understanding of the meanings. A common vocabulary which the agents are committed to has to be specified beforehand. The necessary vocabulary and the communicative rules are defined within the communication protocol. The interacting agents must engage to the same protocol else the communication will prove impossible.

### 2.1.2 Agent negotiation

As negotiation or bargaining one can think of any kind of communication process between two or more entities which aim to achieve an agreement. Another definition given by [Zhang et al., 2005] is that negotiation is the process by which the agents reach a joint decision. A common perception of the negotiation process is that the participants are governed by conflicting goals and each participant makes offers that achieve its goal. But negotiation is not only about conflicting goals since there is also the possibility that the negotiators share a common goal.

Since the very first introduction of multi-agent systems, the agent community has shown great interest in the negotiation process. The existence of multiple agents in the same environment creates the need for a mechanism that will enable these agents to communicate and exchange information in such a way that their goals are achieved. Based on [Jennings et al., 2001] a negotiation process is categorized into three models:

- Game theoretic models, which are mostly concerned with strategic analysis.
- Heuristic models, which are mostly concerned with finding the optimal solution strategy that will give the most preferable outcome.
- Argumentation based models, which give the participating agents the opportunity to provide arguments in order to justify their attitude and persuade the other participants to change their stance.

Another categorisation is given by [Lomuscio et al., 2000] who refers to the computational and to the theoretical models of the negotiation. The computational model provides the structure for the negotiators and it can be thought of as the result of merging the heuristic and the argumentation based models into a single one. The theoretical models provide the foundation for description, specification and reasoning about the negotiation participants.

However, most negotiation processes consist of three phases: the beginning phase, the middle phase and the ending phase. In the beginning phase an agent prepares itself

for the negotiation process, usually through planning. In the middle phase the agent seeks for solutions based on the offers that have been made, usually by searching for the best strategy to follow. Finally, the end phase is concerned with the final details of the negotiation, the obligations of the participating agents and the implementation of the agreement.

Agent negotiation can take many forms depending on the number of the participating agents. A *one-to-one* negotiation takes place between two agents; *one-to-many* is the negotiation between one to many agents with a good example being the auction. Finally there is the *many-to-many* negotiation which corresponds to a distributed negotiation between many agents.

Negotiation varies depending on the agents' goals: another conceptualisation of the negotiation depends on the negotiation. As an example the negotiation kinds can be categorised into task oriented, source allocation or even to the simple negotiations that concern the price of a good. In this project we are dealing with the task oriented negotiation domain: the agents aim to achieve an agreement over a task allocation. The task Oriented Domain (TOD) was introduced by [Zlotkin and Rosenschein, 1993] as an effort to classify the problems that concerned with task sharing and task allocation among autonomous agents. In more detail the domain is the set of the interacting agents and the set of the negotiating tasks. The aim of this negotiation is to achieve an agreement over task allocation or sharing of the initially allocated tasks. Based on this description of negotiation several protocols have been proposed with the Contract Net Protocol [Smith, 1980] setting the basis: an agent advertises a task to the other agents and waits for offers.

In general, negotiation can be considered as the union of: the negotiation set, the protocol, and in some cases the strategies set. The negotiation set includes all the possible proposals that occur in a specific negotiation domain. The protocol specifies all the legal actions of the negotiation participants, the negotiation rules, the acceptable proposals as well as the necessary vocabulary for the negotiation. More specifically, the negotiation protocol must capture and present all aspects of the negotiation process. However it is essential that the protocol does not violate the policies of the agents while it provides them with the freedom of choosing their actions based on their own beliefs and preferences. Lastly, strategies formalise the sets of proposals that each agent makes in order to achieve an agreement to its best interest.

## 2.2 Ontologies

The term ontology comes from the Greek word *ontologia* with the original meaning being the philosophical study of existence. Ontologies have been utilised in a lot of fields and several efforts have been made to provide a formal definition. The most cited definition is the one given by [Gruber, 1993]

”an ontology is a formal explicit specification of a shared conceptualisation”.

The notion of ontology is broadly used in both Philosophical and Information Sciences for describing of what exists and the relations of what exist. More formally it is used to provide a description of a specific domain, the concepts of this domain and the relations between them. In more detail, ontologies provide a classification of a domain in terms of *classes* or set of classes, *properties* of these classes and of the *relations* between the classes, concluding to a complete structured domain.

Artificial Intelligence is another area that has shown a great interest in ontologies. As [Chandrasekaran et al., 1999] states, the term has been adopted to describe two related things: the specific vocabulary that is used to describe a domain and the specialised knowledge about a domain. The first one intends to capture the conceptualisation of the terms used to describe this domain. Although the second definition is closely related to the first one, it is usually used to refer to the whole body of commonly accepted knowledge about a domain, which utilises the vocabulary as a means of representation. Ontologies can also be thought of as content theory because they aim to capture the classification of a domain and the relations that may exist; for example the work done in [Hayes, 1985].

Apart from the conceptualisation of a domain that ontologies offer, they have also attracted the interest of Artificial Intelligence and Computer Science because they can be presented in machine processable languages. In this way structured knowledge can be readable by machines resulting in automated information sharing and reuse. This property of ontologies is the central requirement for communication in distributed systems, where there is an intensive need to exchange information in a form that can be interpreted and reasoned about. For instance, when two services communicate they exchange bits and bytes, while ontologies can enrich this communication and the domain of the discourse. Ontologies apart from data exchange have been also utilised for data integration: merging information is more efficient when the knowledge is formed in a sharable and acceptable format. For all these reasons, ontologies constitute the backbone of agents domain knowledge. Ontologies are used to describe an agent’s

perceptions of the environment it exists, in the domain of knowledge as well as the aims and the needs of this agent. In addition the vocabulary that each agent can use for communication purposes is described on its ontology.

The Semantic Web [W3C, 2005] which is defined as the *"common framework that allows data to be shared and reused across application, enterprise, and community boundaries"*, is another area that has benefited from ontologies.

### 2.2.1 Ontology evolution

Constantly changing environments such as the business industry, create a need for continuously adapting ontologies. Hence, ontologies should cope with these changes in order to meet the environmental needs. Ontology evolution refers to the continuously alteration of the ontologies and to the management process of these alterations as [Haase and Stojanovic, 2005]. The approaches of ontology evolution can be categorised based on the way the changes take place. The first category deals with the users involving to the alteration of the ontologies, [Klein, 2004, Noy et al., 2006]. The second category deals with dynamically updating and learning ontologies without much focus on managing the changes, such as [Alani et al., 2006, Novacek et al., 2007]. Ontology evolution is concerned with changing the information that is currently described within an ontology, changing the perspective of the domain, altering the conceptualisation of the ontology or even adding new information.

[Stojanovic et al., 2002] identifies five stages during ontology evolution: the change capturing phase, the change representation phase, the semantics of change phase, the implementation and change validation phase. The considered to be the major tasks of the ontology evolution life cycle.

The concept of ontology versioning arises on the perspective of ontology alteration based on specific needs. Ontology versioning differs from ontology evolution in the way changes are implemented. While ontology evolution refers to the process of ontology modification while its consistency is preserved, ontology versioning refers to the creation and managing of different versions of the same ontology.

## 2.3 Negotiation Ontology

The intensive need for communication in heterogeneous systems led many researchers to represent communication frameworks in a global way that can be adopted by any agent. In this way, any dissimilarities of the internal architecture of the agents are ignored. A negotiation ontology is the field that focuses on enabling the negotiation process among agents that do not agree on their specifications.

Common negotiation processes entail hard coding the agents with all the necessary features that will enable them to participate effectively in a negotiation process. Negotiation ontologies follow a different approach: the protocol is described by an ontology, which is available to all the agents that want to enter a negotiation.

Negotiation ontologies are divided into two categories: negotiation disambiguation ontologies and negotiation protocol ontologies. The first category provides shared vocabularies, and serves the purpose of solving heterogeneity problems between the negotiation participants. Such ontologies are usually designed especially for cases in which agents are using different vocabularies. Negotiation protocol ontologies, on the other hand, automatically generate negotiation protocols that are going to be used by the participating agents. These ontologies are suitable for agents that have been already hard coded with a negotiation protocol, different from the one used in this specific negotiation [Lopes et al., 2008].

A generic ontology was proposed by [Tamma et al., 2002] in an effort to capture different protocols within the same ontology. This ontology is mainly consists of three categories. The first part includes all the entities involved into the negotiation process, the second part includes the negotiation objects and the third part includes the negotiation process itself.

Another approach to negotiation ontology was proposed by [Strobel, 2002]. The ontology is entirely based on the perspective and behaviour of the evolved agents. A negotiation ontology for sharing learning goals, was proposed by [Supnithi et al., 1999]. Moor [Moor, 2005] proposed an ontology for supporting meaning negotiation between agents.

Finally, [Trojahn et al., 2006] follows a negotiation model for structural multi-agent systems. This kind of system is characterised by roles, role relations and agent groups. When an agent adopts a role, that means that this agent must accept an essential set of constraints. These roles are divided into mediators, agents who are responsible for mediating the negotiation process; and to mappers which are responsible for giving

an output between two ontology mappings. Role relations define the relations among agents such as acquaintance, communication and authority. The agent that plays the source role always has authority upon the destination agent.

## 2.4 Automated planning

Automated planning is the research field which is concerned with forming plans for achieving specific goals. Artificial Intelligence has shown great interest in this field since the early 1970's. The main idea is, given a domain and a problem to find the necessary sequence that will provide a solution to this problem. Automated planning is not suitable in every domain. It is suitable in cases in which

- the actions are complicated,
- actions impose risk, and
- when a new situation is addressed.

Planning can take many forms depending on the action set that is used. Some examples are the path and motion planning, the perception planning and the communication planning. Path and motion planning deals with synthesising of a geometric path of actions and specifying the state variables in the configuration set. Perception planning involves sensing actions for gathering information. Another form of perception planning is the data gathering planning which uses a system querying process instead of sensing. There is also communication planning which is concerned with cooperations issues among agents.

The planning domain can either be specific or independent: specific domains such as *STRIPS* [Farquhar et al., 1996] use specific representations and techniques which are adapted to the problem, while independent domains use generic representations and techniques. Representation stands for the way a planning problem is described. Based to [Ghallab et al., 2004] it is categorised into set-theoretic representation, classical representation and state-variable representation. The first represents the states of the world as a set of propositions and the actions as syntactic expressions of preconditions and effects. The preconditions specify the propositions of the state that must be true for the action to be applicable. The effects specify the propositions that are true after the action is applied. In classical representation the actions and the states are represented by first order literals and logical connectives. Finally in the state-variable

representation the states are represented by tuples  $\{x_1, \dots, x_n\}$  and the states as functions that map each state tuple to another state tuple.

## 2.5 Ontology Repair System

The Ontology Repair System (ORS) is a plug-in tool for dynamic ontology repair in a planning context. It facilitates communication among a planning agent (PA) and a service providing agent (SPA) in cases of ontological mismatches. It consists of three main parts: the Translation System, the Diagnostic System and the Refinement System. In Figure 2.1 the interactions between the planning agent (henceforth PA), the ORS and the service providing agents (henceforth SPAs) are shown.

The Ontology Repair System and the agent communication system are implemented in *Sicstus Prolog* [Sicstus, 2005], which by itself creates the need of modifying the outcome of the planning procedure into an understandable format for the system. This requirement results to the translation system of ORS. The translation system is responsible for translating the agent's KIF ontology into a Prolog ontology for the planning and for the communication purposes. This translation is one way.

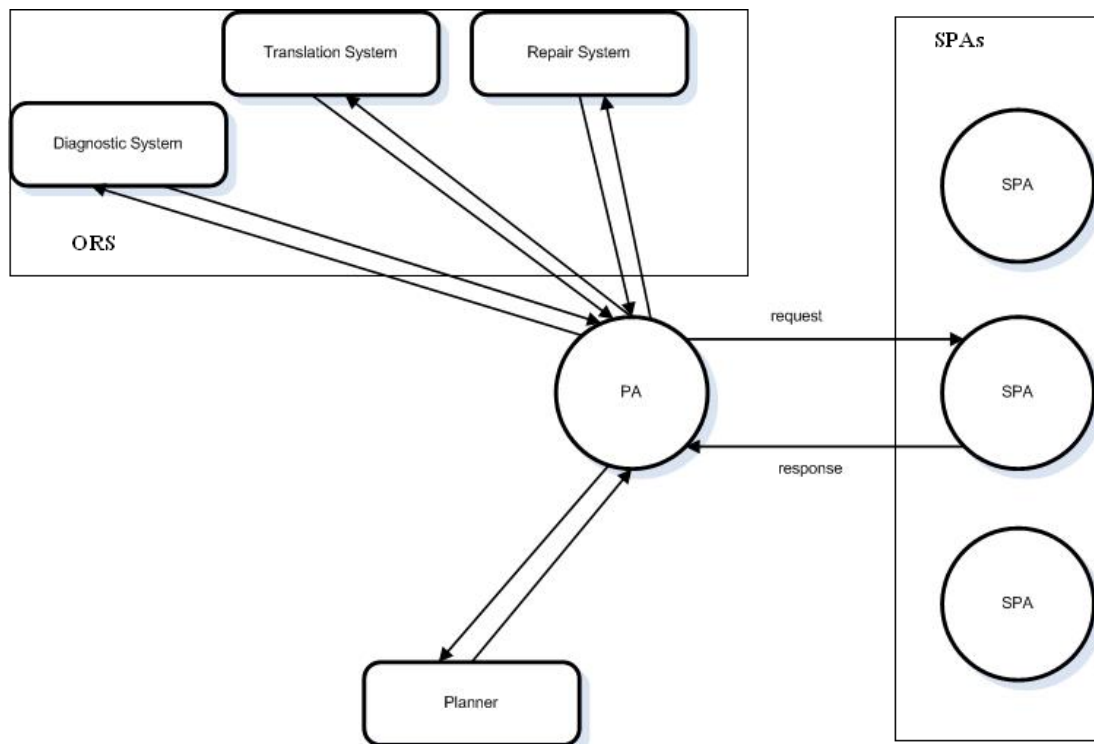


Figure 2.1: The interactions between the PA, the ORS and the SPAs

The PA has access to a planner, which is responsible for composing the plan based on the predefined goal. After the plan steps and the preconditions are justified, the plan is sent back to the PA in order to start execution. During plan execution, the PA interacts with one SPA at a time, and requests to perform actions included in the plan. When the PA requests from the SPA to perform a task, the SPA firstly checks that it can perform this task and then the preconditions of this task. These preconditions are classified in the preconditions that can be checked independently and to the preconditions that need to be checked with the PA. The only way to confirm that these preconditions match with the PA is through dialogue. So, the SPA questions the PA about these preconditions.

A communication failure between the PA and the SPA can occur because of different reasons. A possible failure occurs if the SPA is requested to perform a task which is not capable of. In case that one of the preconditions of the requested task is not satisfied, the communication fails. Another factor that causes communication failure is a possible disagreement about the precondition set. If the SPA sends a question to the PA, which the PA does not recognise then communication fails. Such questions are characterised as *surprising questions*

The Diagnostic System is utilised when a communication failure occurs, to find the exact cause of the failure [McNeill et al., 2004, McNeill and Bundy, 2007a]. Since the exchanged messages during agent communication represent some parts of the agents' ontologies, the diagnostic system takes as inputs this message and performs an exhaustive check on the PA's ontology to find any possible mismatches. In more detail, it refers to the PA's plan step justification, which contains the rules of this step, to find the general area the error occurs. For instance it will check whether the preconditions match with those of the SPA. However the communication does not always fail due to ontological mismatches, but there are other cases such as the case of requesting a task from an incorrect SPA: the SPA cannot perform this task. For this reason, the Diagnostic System also checks whether the PA has requested a task from the correct SPA.

In ORS only the interaction between the PA and the SPA is concerned with with. It is assumed that the agents are helpful and willing to perform the requested tasks. Possible problems that may occur from the PA's and SPA's interaction with third parties are ignored so that the system is simpler. Thus, the PA's and SPA's ontologies cannot be changed by interactions with external agents. In this way, a task refusal means that either the preconditions are not fulfilled or that the SPA has been requested to perform

a task that is incapable of. ORS deals with ontological mismatches of the same type. The mismatches that are diagnosed are concerned with:

- **Predicates.** The same predicate, in the PA's and the SPA's representation, differs on the generalisation level or on the arity: the predicate's name is more general and/or the number of the arguments is greater than in the other representation.
- **Arguments.** The same argument, in the PA's and the SPA's representation, differs on the domain: in one of these two representations the domain of the argument is more general than in the other.
- **Preconditions.** The preconditions of the PA differ from the preconditions of the SPA: a precondition is missing.
- **Class arguments:** in the PA's representation the class of arguments is different from the SPA's, or the arguments are transposed. Another mismatch occurs when in the PA's representation a class has different name for the SPA's.
- **Class hierarchy.** The class hierarchy differs: the PA's representation has different superclass from the SPA's.
- **Individuals:** they have different name or they belong to different classes.
- **Facts:** in the PA's representation a fact differs from the SPA's.

After the Diagnostic System finds the specific mismatch that caused the failure, the Refinement System is responsible for performing the repair that will align these two ontologies. Currently, ORS is a plug-in only for the PA so the repair will always be performed in PA's ontology. The repair process is based on abstraction and refinement techniques. An ontology can be generalised, by removing details, or specialized by adding details. These two techniques are used for altering predicates and action rules within the ontology. A predicate can be altered by making its name more general or more specific. Additionally abstraction or refinement can be used to add or remove arguments. Action rules are changed by altering the preconditions and/or the effects of an action by using these two techniques. Nevertheless, repair is also concerned with classes, individuals and facts. In order to alter these attributes, other techniques apart from abstraction and refinement are used. First of all, the class arguments can be changed by linking the new predicate occurrence to the existing one, and then finding the class of the new argument either through examination of the ontology or through

questioning the agent. Secondly, the position of the the arguments in a class can be switched. Another repair that deals with classes is changing the name of a class or the or the hierarchy of the classes. An individuals can be repaired by changing its name or the class it belongs. Facts can only change based on agent's beliefs, and finally whole ontological objects can be added or removed [McNeill and Bundy, 2007b]

Another feature of ORS is that it deals with ontology protection [Bomersbach, 2011]. Ontology protection aims to restrict prevent specific ontological parts from any alteration and depends on the application area of the ontology. For instance protection can represent safety, trust or even financial issues. Ontological protection implies that before any kind of refinement is performed, we have to make sure that there does not exist any kind of protection at this ontological part. The protection levels that are possible are high and low. So, since the exact ontological mismatch has been identified, firstly it is important to confirm that this repair does not concern a protected ontological part. In case the part to be repaired does not have any kind of protection, it is sent to the Refinement System with all the necessary information for the repair process. After the mismatch is corrected, the PAs ontology is updated and then the system continues with plan execution.

Work conducted by [Bomersbach, 2011] introduced to the system a simple negotiation process for the suggested repairs. The negotiation process is based on ontology protection: if the PA's ontological part that needs repair is protected, then the negotiation begins. If this ontological part is not protected, then the PA will make the repair without negotiating it. The protocol that is followed by this negotiation system, defines two permitted messages that can be exchanged during the negotiation: *repair* and *repair2*. Both messages can only be sent by the PA. *Repair* message is sent by the PA to start the negotiation: the PA informs the SPA about the repair and requests to perform it. Since the SPA receives this message, it checks its ontology for protection and it obtains the protection level. It will accept to perform the repair only if it has no protection; otherwise it will refuse. If the PA receives a negative response to the *repair* message and the protection it has is low, it will perform the repair. Otherwise it will send the *repair2* message; it requests again from the SPA to perform the repair. This time, if the SPA has low protection it will agree to perform the repair; otherwise it will refuse and the plan will fail. The outcome of this protocol is that the PA always performs the repair, unless it has high protection. In this case the SPA will perform the repair if it does no have high protection. If both agents have high protection, then none of them will perform it and the plan will fail.

# Chapter 3

## Negotiation Protocol

### 3.1 Introduction

One of the goals of this project is to define a negotiation protocol under the scope of ORS to enable agent negotiation for ontological repairs. The protocol provides the vocabulary, the set of negotiation rules, and the set of legal agreements that the agents will use as a basis to establish a successful negotiation. After a communication failure between the PA and the SPA, the PA obtains from ORS the diagnosis about the exact reason that caused it, as well as the suggested repair based on this diagnosis. In the original ORS, the PA agreed to this repair and performed it, while the SPA never altered its ontology. The protocol that we define enables the PA and the SPA to negotiate over the suggested repair, and to reach an agreement over which of these two agents is going to alter its ontology. The process that is followed before the negotiation starts, is that ORS suggests a repair as well as the specific ontological part the repair must be performed on. Based on this suggestion both the PA and the SPA check their ontologies for any kind of protection. The protection level they obtain is the key to the negotiation that will follow; the protection of ontological parts aims to represent the cost of an alteration at these parts, while the protection levels show, in a way, the cost levels. As an example, an ontological part that is characterised with high protection declares that in any case, this part must not be altered and that any alteration is considered harmful for the agent. However, the way protection captures risk or vary on the application area of each ontology. So, we also consider cases in which the cost of the communication failure is greater than the cost of an ontological alteration. On the other hand an ontological part without any kind of protection means that any alteration would have less side-effects to this agent. So protection can be thought of

as expressing the cost of an alteration in a specific ontological part. Based on the protection level the agents obtain, they form their preferences and these preferences will generate each agent's negotiation strategy.

In order to provide these two individual agents with all the essential knowledge that will let them interact successfully during the negotiation process, we define a protocol for the *one to one* (PA-SPA) negotiation that will enable the agents to come to an agreement about which one should perform the suggested repair. The negotiation protocol defines the negotiation in terms of:

- the negotiation participants,
- the negotiation object,
- the permitted offers,
- the rules that govern the negotiation process,
- the conditions of the interactions between the agents,
- the permitted agreement,
- the obligations that entails an agreement, and
- the vocabulary, (the permitted exchanged messages).

## 3.2 Negotiation Domain

The negotiation begins after ORS diagnoses the cause of of the communication failure, and suggests the essential repair. Since the aim of the negotiation is to conclude which agent is going to perform the suggested repair, the domain of this negotiation is task allocation. However, ORS suggests only one repair for a specific ontological part per communication failure, so consequently we restrict the number of the possible negotiable tasks per negotiation round to be exactly one. As a negotiation round, we define an established negotiation between the PA and the SPA that has successfully completed (reached its terminal state).

In our system, we believe that a successful communication between the PA and the SPA is fruitful for both agents. First of all, the PA can continue performing its plan without any failures, while the SPA (in real life applications) can be rewarded for its performed actions by the PA. As an example, when the PA asks from an *electricity*

*provider* agent for electricity supply, the *electricity provider* agent will perform the requested action, and as a reward the PA will pay an amount of money. In some other cases, the inability of the SPA to perform an action for another agent might conflict with its own beliefs and desires while at the same time this could imply financial loss. For reasons such as these, we assume that both agents aim to establish a successful transaction, so in cases of communication failure they both aim to fix the cause of this failure. Under this assumption, agreement incurs the maximal benefit for both agents.

### 3.2.1 Negotiation Parties

Because the agents participating in the negotiation are the same agents between which the communication initially failed, the number of agents that can participate in the negotiation is restricted to *two*. Each of them is assigned with a specific role for the negotiation round that is going to follow. The agent that starts the negotiation, in our case the PA, is assigned the role of the *initiator* and the agent that receives a call to participate in the negotiation, the SPA, is assigned the role of the *participant*.

Besides the negotiators, we introduce another kind of agent: the expert agent. The expert agent does not participate in the negotiation process. Instead, it is assigned an advisory role. The main responsibility of this agent is to advise the agents about the ontological representation it considers to be the correct one (see section 3.2.6).

To sum up, the set of the maximum agents that are presented during the negotiation process, is the set

$$A_g = \{initiator, participant\}$$

The maximum number of participating agents being *two*, is justified by the negotiation process that we are dealing with being an *one-to-one* negotiation. However, the protocol provides the flexibility of defining more participants for a *many-to-many* negotiation, (see future work).

### 3.2.2 Negotiation Object

The negotiation process aims to enable the PA and the SPA to negotiate over which one is going to perform the suggested repair by ORS repair. Therefore, the set of negotiation objects, which contains all the possible matters the agents negotiate over that are defined for this domain, should be constituted by the set of all possible repairs ORS suggests. Every time communication between the PA and the SPA fails, ORS takes as input the PA's ontology and the exchanged messages between these two agents

before the communication collapsed, diagnoses the reason of the failure, and suggests a repair. This repair is suitable only for the PA though, since it is suggested based on the PA's ontology. Consequently the repair that the SPA should perform differs from the original suggestion: it is the inverse one. To illustrate this point, consider the case that a communication failure is caused because the corresponding arguments on the PA's and on the SPA's ontology differ in arity: the PA's ontology has one more argument than the SPA's, so, the suitable repair for the PA would be the propositional refinement repair: one argument is removed from the predicate. Now, if the SPA performs the same repair, this repair leads to raising the number in the predicate. For this reason, to resolve the issue of the different predicate arity, either the PA should remove an argument from its predicate or the SPA should add the necessary argument to its predicate (propositional abstraction).

However, not all the repairs have an inverse one which is going to be performed by the SPA. For instance when the repair deals with negating a PA's postcondition, there does not exist an inverse repair that unnegates the corresponding SPA's postcondition. Taking into account all these issues, the *Negotiation Object* can be decomposed to *seven* repairs:

1. predicate refinement
2. predicate abstraction
3. propositional abstraction
4. propositional refinement
5. domain abstraction
6. domain refinement
7. switch arguments

Apart from the negotiation domain, it is essential to consider the domain on which the repair is made; It is not possible to define a repair as a negotiation object without providing all the essential information about this repair. One of the basic reasons that the essential information for this repair should be provided to the negotiation parties is that this information is needed by ORS in order for the repair to be performed. Secondly, by providing the essential information in addition to the name of the repair,

we are creating a more flexible protocol in time which is not affected by cases where the repair remains the same but for some reason the name changes.

Under these restrictions we define the negotiation domain as a triplet  $\langle Ag, MT, c \rangle$ , where

- $Ag = \{A_1, A_2, \dots\}$  is the finite set of the agents that can participate in a negotiation stage,
- $M = \{M_1, \dots, M_n\}, n \leq 9$  is the set of all the permitted messages the agents can exchange,
- $T = \{T_1, T_2, \dots, T_k\}, k \leq 7$  is the finite set of all possible negotiation objects where each T is constituted by a tuple  $\{N, I\}$  where  $N$  is the name of the repair and  $I$  is the information regarding this repair, (this is the set of repairs that ORS can suggest) and
- $c : 2^T \rightarrow R^+$  is the cost of performing a task T, which is a positive number.

### 3.2.3 Cost Function and Utility

The implementation we followed in this project deals with situations of ontological protection: each agent's ontology might be constituted of parts characterised as protected. Ontology protection aims to prevent the protected parts from any kind of alteration. The parts of the ontology that can be protected are: the whole predicate, the predicate's arity, the whole argument and the argument's type.

A protection in the whole argument means that no alteration in the predicate's arguments is permitted. This kind of protection denies permission of performing any of the *predicate refinement*, *propositional refinement*, *propositional abstraction*, *domain abstraction*, *switch arguments*, *precondition abstraction*, *precondition refinement* or *postcondition negation* repairs since all these repairs either need to add or to remove an argument, change an argument's type or even switch the positions of the arguments.

In the same way the predicate's arity protection aims to prevent the predicate from repairs that change the arity of its arguments. Such repairs are the *propositional refinement* and the *propositional abstraction*; the first one removes an argument from the predicate, while the second one adds an extra argument. When the whole argument is protected the aim is to prevent any kind of alteration in the argument's type. For this reason the repairs *propositional refinement*, *domain abstraction* and *switch arguments*

are prevented by this type of protection. Finally when the argument's type is protected the repair that is prevented is *domain abstraction*.

So, before each agent participates in a negotiation about any kind of repair it firstly has to check the part of its ontology, the suggested repair deals with for any kind of protection, and then enter the negotiation. The level of protection that a part of an ontology can have differs: the part can be marked with high protection, low protection or no protection at all. High protection indicates that an alteration of this part can cause damage to the agent, and makes sure that no alterations is taken place. Low protection, does not restrict an alteration at the same degree as high protection, Protection is more loose and allows the agent to perform an alteration, but also declares that any kind of alteration is risky [Bomersbach, 2011]. Finally, no protection allows the agent to alter this ontological part.

Based on the protection levels, we define as  $cost_i(T_k)$  the cost that an agent  $A_i \in Ag$  has of performing the task  $T_k \in T$  as a positive value within the range  $[0, 2]$ . The space  $[0, 2]$  comes from the protection levels that can occur in an agent's ontology, mapping the value 0 to highly protected and the value 2 for unprotected parts. As an example, we consider that the cost that an agent  $A_i \in Ag$  has for performing the task  $T_k \in T$  is zero, if and only if the part of its ontology that the repair is about has no protection. In the tables 3.1, 3.2, 3.3, 3.4. 3.5 we present the levels of protection and the corresponding cost values.

In order to achieve a negotiation, each agent should have a preference about this repair, which is going to form the agent's strategy during the negotiation process. Each agent's preference is expressed by the *utility* function which gives an estimation of the effect each repair will have on the agent's ontology. The utility is given by the

whole predicate protection	no protection	low protection	high protection
predicate refinement	$cost = 0$	$cost = 1$	$cost = 2$
predicate abstraction	$cost = 0$	$cost = 0$	$cost = 0$
propositional abstraction	$cost = 0$	$cost = 1$	$cost = 2$
propositional refinement	$cost = 0$	$cost = 1$	$cost = 2$
domain abstraction	$cost = 0$	$cost = 1$	$cost = 2$
domain refinement	$cost = 0$	$cost = 0$	$cost = 0$
switch arguments	$cost = 0$	$cost = 0$	$cost = 0$

Table 3.1: Cost values for whole predicate protection

predicate's arity protection	no protection	low protection	high protection
predicate refinement	$cost = 0$	$cost = 0$	$cost = 0$
predicate abstraction	$cost = 0$	$cost = 0$	$cost = 0$
propositional abstraction	$cost = 0$	$cost = 1$	$cost = 2$
propositional refinement	$cost = 0$	$cost = 1$	$cost = 2$
domain abstraction	$cost = 0$	$cost = 0$	$cost = 0$
domain refinement	$cost = 0$	$cost = 0$	$cost = 0$
switch arguments	$cost = 0$	$cost = 0$	$cost = 0$

Table 3.2: Cost values for predicate's arity protection

whole argument protection	no protection	low protection	high protection
predicate refinement	$cost = 0$	$cost = 0$	$cost = 0$
predicate abstraction	$cost = 0$	$cost = 0$	$cost = 0$
propositional abstraction	$cost = 0$	$cost = 0$	$cost = 0$
propositional refinement	$cost = 0$	$cost = 1$	$cost = 2$
domain abstraction	$cost = 0$	$cost = 1$	$cost = 2$
domain refinement	$cost = 0$	$cost = 0$	$cost = 0$
switch arguments	$cost = 0$	$cost = 0$	$cost = 0$

Table 3.3: Cost values for whole argument protection

whole argument protection	no protection	low protection	high protection
predicate refinement	$cost = 0$	$cost = 0$	$cost = 0$
predicate abstraction	$cost = 0$	$cost = 0$	$cost = 0$
propositional abstraction	$cost = 0$	$cost = 1$	$cost = 2$
propositional refinement	$cost = 0$	$cost = 0$	$cost = 0$
domain abstraction	$cost = 0$	$cost = 1$	$cost = 2$
domain refinement	$cost = 0$	$cost = 0$	$cost = 0$
switch arguments	$cost = 0$	$cost = 1$	$cost = 2$

Table 3.4: Cost values for whole argument protection

argument's type protection	no protection	low protection	high protection
predicate refinement	$cost = 0$	$cost = 0$	$cost = 0$
predicate abstraction	$cost = 0$	$cost = 0$	$cost = 0$
propositional abstraction	$cost = 0$	$cost = 0$	$cost = 0$
propositional refinement	$cost = 0$	$cost = 0$	$cost = 0$
domain abstraction	$cost = 0$	$cost = 1$	$cost = 2$
domain refinement	$cost = 0$	$cost = 0$	$cost = 0$
switch arguments	$cost = 0$	$cost = 0$	$cost = 0$

Table 3.5: Cost values for argument's type protection

difference between the cost each agent has of not performing the repair and the cost of performing it. More formally, we define the  $utility_i(T_k)$  of an agent  $A_i \in Ag$  for the repair  $T_k \in T$  as  $utility_i(T_k) = cost(norepair) - cost(T_k)$ . By defining the utility in this function, we provide the agents with the freedom to chose of not performing the repair. However, this implies the risk of both agents choosing this strategy, leading the system to failure. In this project we considered the cases that both agents are willing to perform the repair, but we do not suggest that this is the only way the negotiation can proceed.

Taking into account that all the agents aim to complete their initial communication successfully, which implies that the repair must be performed by one of these agents, we consider the cost of each agent for not performing the repair to be equal with the highest cost a repair can have for an agent: 2. Based on these facts each agent's utility is given by:  $utility_i(T_k) = 2 - cost(T_k)$ .

Moreover, we define the social welfare of an agreement to measure the utility this agreement has for the system in total. The system's total utility is the sum of the utility each agent within the system has for this agreement. More formally, the social welfare for an agreement  $\delta_j(T_k)$  of agent  $j$  performing the repair  $T_k$  in a system of  $Ag = \{1, 2\}$  agents, is given by:

$$social\ Welfare(\delta_j(T_k)) = \sum_{i=1}^2 utility_i(\delta_j(T_k))$$

After an agent checks its ontology for protection, it obtains the protection level and then it forms its utility for this repair. Based on this utility, the agent will decide whether or not it will enter the negotiation or not. The value of the highest utility an agent can have is two, in case of no protection at the agent's ontological part the repair

refers to. The lowest value that an agent's utility can have is *zero* when the part of its ontology is characterised by high protection, while its utility can be *one* when the protection is low. Since the agents act within a cooperative system and they both aim to complete their transaction which was interrupted by a communication failure, they will enter the negotiation in all cases that their utility is greater than zero. During the negotiation both agents aim to come to an agreement that maximises the social welfare.

### 3.2.4 The Negotiation Offer and the Negotiation Strategy

In the negotiation domain we define, the only possible agreement that can be reached is the allocation of a repair in one of the two participating agents, while in each negotiation round agents negotiate over a single repair. Thus, the search space of the possible offers an agent can propose is greatly restricted: in a system containing  $n$  agents, the number of the possible offers  $Z_i$  an agent  $i$  can consider is  $|Z_i|(n - 1)$ . Our negotiation domain is consisted of *two* active participating agents, so the possible offers an agent  $i$  can make are  $|Z_i|(2 - 1) = 1$ , and this is the offer of performing the repair. The agents can always chose not to perform the repair, but this is not considered as an offer.

The strategy the agents follow during the negotiation process is quite straightforward: they use their protection levels to form their utility for the suggested repair, and based on the value of their utility they decide how risky the performance of the repair would be for them. The definition of *risk* we use, is the level of the potential damage a repair can cause to an agent or an organisation the agent represents. Since both agents aim to continue their previous interaction that was interrupted by a communication failure, they both agree that the agent which should perform the repair should take the least possible risk. If  $utility_i$  is greater than  $utility_j$ , this means that agent  $i$  performing the repair is less risky than agent  $j$  doing so.

From one point of view, the limitation of the possible offers that can occur decreases the computational time for deciding which offer is the most suitable for an agent. Furthermore, since the agents aim to continue their previous communication successfully, the allocation and performance of the repair to one of the participating agents raises the social welfare.

### 3.2.5 Negotiation Agreement

The negotiation domain in which the agents act is a simple cooperative task oriented domain, in which the negotiation agreement that is reached aims to maximise the social

welfare. The agreement is concerned with the allocation of a single task: which agent is going to perform the suggested repair. For this reason the negotiation object is defined as a set  $NegotiationObject = \{T_1, T_2, \dots, T_k\}$  where  $k \leq 7$  (7 is the number of the inversed repairs). A negotiation offer is constituted by an offer an agent makes about performing the repair or not. An offer is done based on the utility each agent has, and is either accepted or refused, again based on the other agent's utility. As mentioned before, an agent  $A_i \in Ag, i \leq 2$  will enter the negotiation and will make an offer for the negotiation object  $T_k \in T, k \leq 7$  if and only if  $utility_i(T_k) > 0$ . The offer the agent  $A_i$  makes is accepted by the other agent, and an agreement  $\delta_i$  of the agent  $A_i$  performing the repair  $T_k$  is reached if and only if the other agent's utility is lower than of the agent  $A_i$ . This is justified by the fact that both agents aim to reach an agreement  $\delta$  which will bring to the system the highest possible welfare. At this point, we should clarify that we do not consider the case that none of the agents performing the repair as an agreement. Instead we think of this case as a failure to reach an agreement.

As an example, let  $utility_i(T_k) = 1$  and  $utility_j(T_k) = 0$  be the utilities the agents  $i$  and  $j$  have for the repair  $T_k$ . In this negotiation round there are *two* possible agreements: The agreement  $\delta_i$  of agent  $i$  performing the repair and the the agreement  $\delta_j$  of agent  $j$  performing the repair. For the agreement the agent  $i$  has  $\delta_i$ ,  $utility_i(\delta_i) = 1$  and the agent  $j$  has  $\delta_i$ ,  $utility_j(\delta_i) = 2$  since there is no repair cost. The social welfare for the agreement  $\delta_i$  is

$$social\ Welfare(\delta_i) = \sum_{k=1}^2 utility_k(\delta_i) = 3.$$

For the agreement  $\delta_j$ , the agent  $i$  has  $utility_i(\delta_j) = 2$  since there is no repair cost, and the agent  $j$  has  $utility_j(\delta_j) = 0$ . The social welfare for the agreement  $\delta_j$  is

$$social\ Welfare(\delta_j) = \sum_{k=1}^2 utility_k(\delta_j) = 2.$$

Consequently the agreement  $\delta_i$  is preferred to the agreement  $\delta_j$  by the agents.

More formally, let  $T_k \in T$  be a negotiation object, and the agents  $A_i, A_j \in Ag$  be the negotiation parties, an agreement  $\delta_i$  of the agent  $A_i \in Ag, i \leq 2$  performing the repair  $T_k \in T, k \leq 7$  is reached between the agents  $A_i$  and  $A_j \in Ag, i \leq 2$  if and only if social welfare( $\delta_i$ ) is maximal, consequently if and only if  $utility_i(T_k) > utility_j(T_k)$ , where  $utility_i(T_k) = 2 - cost_i(T_k)$  and  $utility_j(T_k) = 2 - cost_j(T_k)$  for the agents  $A_i$  and  $A_j$  respectively.

An agreement  $\delta_k$  is said to be at least as good for the agent  $A_i$  as for the agent  $A_j$  if  $utility_i(T_k) \leq utility_j(T_k)$ .

We insist that an acceptable agreement cannot conflict with any of the negotiators preferences, thus we restrict the set of the acceptable negotiation agreements to belong to the set of the positive *utility* values for both agents as shown in Figure 3.1. In addition, since the negotiation object is concerned with the allocation of a single task, the repair, the set of possible agreements in a negotiation round is

$\delta = \{ \text{PA performs the repair, SPA performs the repair} \}$ .

The negotiation agreement that is reached, in every round, is considered as *Pareto optimal* since there is no other possible agreement that could raise one agent's utility without making the other agent worse off. Let  $A_i$  and  $A_j$  be the two negotiation participants,  $T_l$  be the negotiation object and  $utility_i(T_l) = 1$ ,  $utility_j(T_l) = 0$  the agents' utilities of performing the repair  $T_l$ . Based on the agreement definition within our protocol, the agreement that is reached between these two agents is the  $\delta_i$  of agent  $A_i$  performing the repair, since  $utility_i(\delta_i) = 1$  and  $utility_j(\delta_i) = 2$ . This agreement is Pareto optimal because the only alternative agreement  $\delta_j$  of agent  $A_j$  performing the repair increases the agent's  $A_i$  utility from *one* to *two*, but decreases as well agent's  $A_j$  utility from *two* to *zero*.

### 3.2.6 Expert Agent

Besides the negotiation participants, we introduce another agent within our system which does not have an active role in the negotiation process. This agent plays the

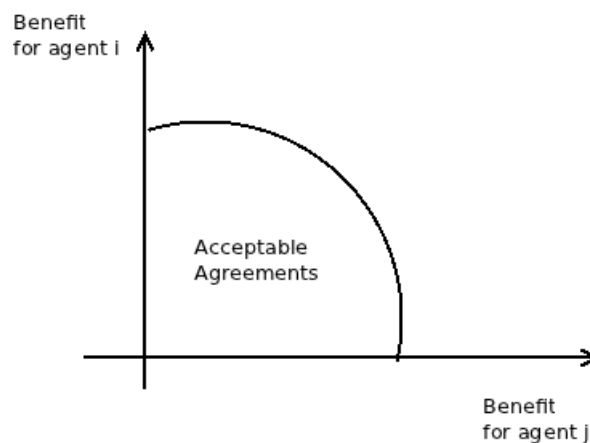


Figure 3.1: The set of the acceptable agreements

role of the *expert* agent. The main responsibility of the expert agent, is to advise the agents over which one should perform the repair based on its domain of expertise. The knowledge of this expert agent is not abstract but specific in the problem domain, and the two negotiation participants take advantage of this knowledge by consulting its opinion about which agent has the correct representation.

The existence of this agent is justified by the fact that in several cases the agents are not able to reach an agreement based only on their own knowledge, so a suggestion by a third agent would be considered as advantageous for reaching a negotiation deal. This expert agent is not considered as part of the negotiation domain, since its existence in the system is not guaranteed. A possible occurrence of this agent is during a negotiation round in which the agents are not able to reach an agreement. As an example, the expert agent can occur during a negotiation phase in which both participating agents want to perform the repair. At this point, the participating agents can ask for the expert agent's opinion and after they obtain it, they can decide whether they will follow it or not. As an example, if the ontological mismatch among the negotiating agents concerns which of *pounds* or *sterling* is the correct representation, the expert agent would have expertise knowledge on the domain *currency*. This is a typical case of when the expert agent is utilised by the system.

For the purposes of our project, we placed the expert agent at the same environment the negotiators act. However, an expert agent can come from other systems as well.

The expert agent's opinion can be considered as motivation for the suggested agent to perform the repair, however we remain faithful to the agents' autonomy and we do not force them to follow this opinion. The expert agent has an advisory role: that is, it is not compulsory for the negotiators to follow the expert's recommendation. Rather, the expert's opinion is used to modify the utility of other agents and aid the progress of the negotiation process. In addition, the expert agent's opinion can provide feedback to each agent about its ontological representation which can be used by each individual as an evaluation of its knowledge mean.

### 3.3 Negotiation Process

The negotiation domain is a task oriented domain which by its nature restricts the negotiation process into being simple with straightforward rules. The negotiation object can only be the suggested repair from ORS. The idea we follow is simple: we have

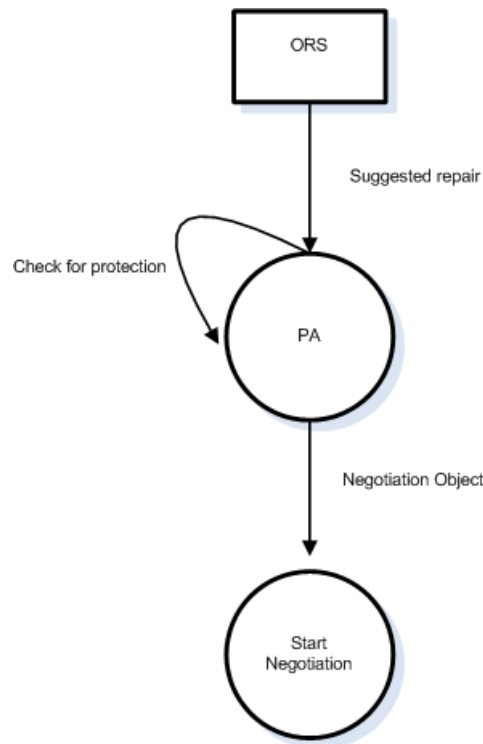


Figure 3.2: Pa starts the negotiation

designed a protocol that enables the agents to negotiate over which one is going to perform the repair with respect to their utilities, their autonomy and the repairs that ORS can perform. Bearing in mind that ORS can perform only a single repair at a time, the offers that can occur during the negotiation process are limited; the agents cannot share the tasks neither can they exchange the tasks they have been allocated. There is only one repair to negotiate, which by the end of the negotiation must be allocated to one of the two participants. This fact limits the set with the permitted negotiation offers to have only *one* offer: the offer of performing the repair. Secondly, we restrict the number of the negotiation participants; since initially the communication failure occurs between two agents, the PA and the SPA, the number of the negotiation participants can only be exactly *two*. Lastly, we have to take into account the possible protected parts of each agent's ontology, which aim to prevent these parts of any alteration.

Based on the restrictions that we are facing, it seemed reasonable to use the contract net protocol [Smith, 1980] as a basis for this negotiation. The contract net protocol follows a simple idea: an agent sends a call of proposals for a specific task to all the other agents and waits for offers. This action starts the negotiation. In the case that an agent is interested in performing this task, it sends back to the agent (the initiator of

the negotiation) a proposal for this task; otherwise it sends a refusal to participate into the negotiation. In case that a proposal has been made, the initiator checks whether this proposal satisfies it and either accepts or rejects it. Using this protocol as a basis, we built on ORS.

Before the negotiation starts, the PA has obtained the suggested repair and has already checked the part of its ontology for any kind of protection. Based on this protection it has formed the utility for this repair. The agent that first enters the negotiation is the PA, which plays the role of the *initiator* during the negotiation round that will follow. The PA begins the negotiation, by sending a *call for proposal* for the inverse repair of the initial one suggested by ORS to the SPA as shown in Figure 5.1. The SPA has the role of the participant during the negotiation, and after receiving the *call for proposal*, it sends back to the PA a message declaring that it has received it. Depending on its utility, the SPA will act accordingly Figure 3.7.

In case  $utility_{SPA}(T_k) = 0$ , the cost is considered to be higher than the benefit of performing this repair while the risk it takes in case it decides to perform it is very high. Consequently it will refuse to participate in the negotiation by sending back to the PA a *refusal* for the *call for proposal* it has previously received. Then the SPA leaves the negotiation, which leads to the termination of this round. This kind of negotiation round is considered as successful with respect to the agents' freedom to enter or leave the negotiation process on demand. The negotiation outcome of this round is that the PA will perform the repair, even if an official agreement has not been reached as shown in Figure 3.3.

In case  $utility_{SPA}(T_k) > 0$ , then the benefit is greater than the cost from performing the repair, so it enters the negotiation process by sending back to the PA a *proposal* for performing the repair that the *call for proposal* was about, along with the value of its utility as an agreement for this action. After the PA receives the *proposal*, it compares the value of its own utility to the one of the SPA, and acts accordingly. In

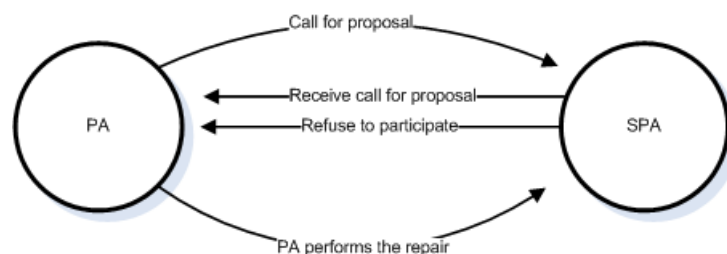


Figure 3.3: SPA refuses to participate in the negotiation

case  $utility_{PA} < utility_{SPA}$ , this means that the PA's utility is not as high as the SPA's, so the PA takes more risk if it performs the repair than the SPA would. Consequently, the PA accepts the proposal and the agreement that is reached is that the SPA will perform the repair, as illustrated in Figure 3.4.

If  $utility_{PA} > utility_{SPA}$ , the SPA's utility is not as high as the PA's, so this means that the PA takes less risk if it performs the repair than the SPA does. So, the PA prefers to perform this repair which maximises the social welfare and would possibly harm SPA. For this reason, the PA rejects the *proposal*, by sending back to the SPA a rejection message, and this negotiation round comes to its end. The agreement that it reached is that the PA will perform the suggested by ORS repair. See also Figure 3.5.

The fourth situation that might occur is when  $utility_{PA} = utility_{SPA}$ . In this case, the *expert* agent is utilised to provide a suggestion about which agent it considers as the most suitable candidate for performing this repair. After the expert agent's suggestion and if and only if the initiator and the participant agree to accept this suggestion, the utility of the suggested agent is increased by *one*, and then the agents act according to this utility change, see also Figure 3.6.

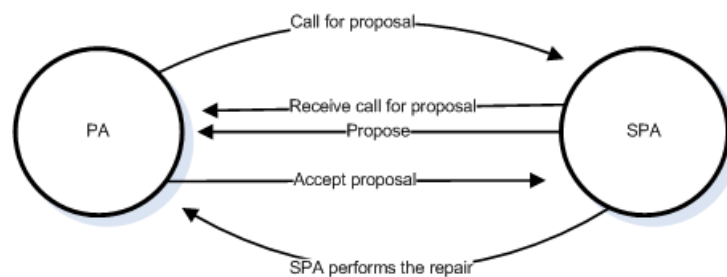


Figure 3.4: Agreement reached: SPA performs the repair

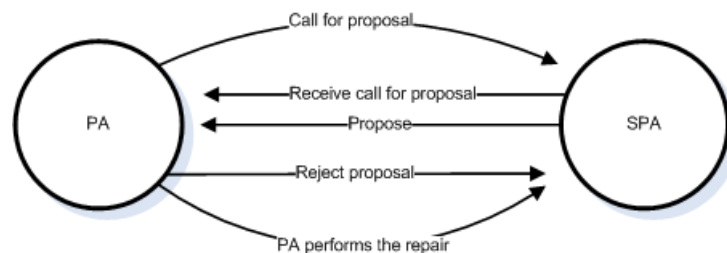


Figure 3.5: Agreement reached: PA performs the repair

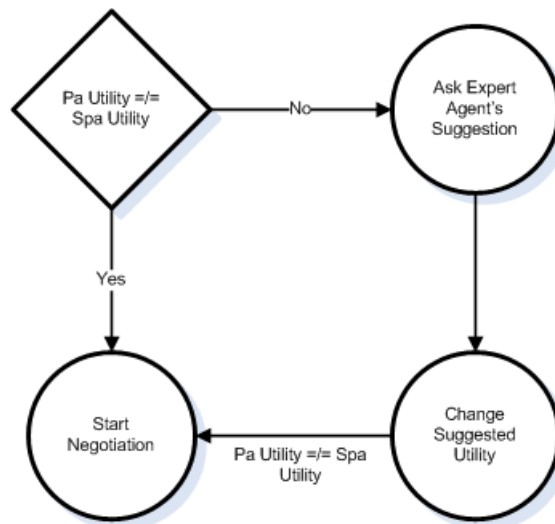


Figure 3.6: Consulting the expert agent

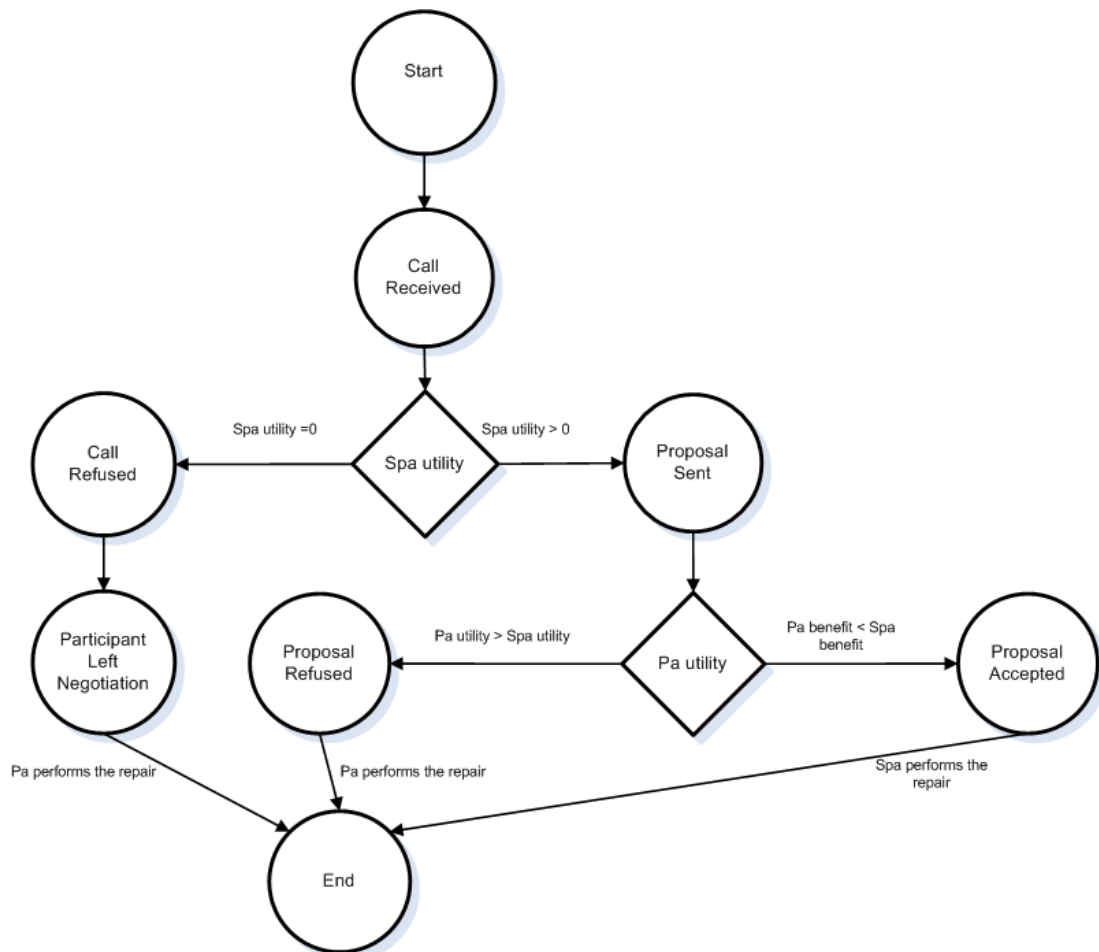


Figure 3.7: The negotiation phases

### 3.3.1 The negotiation messages and the corresponding negotiation phases

In order to provide all the necessary means for the agents to reason about the negotiation process and to clarify the stages of the negotiation, we define the set of all possible negotiation messages and their corresponding phases. For the purposes of the negotiation process, we define three categories of permitted messages that can be exchanged during the negotiation process: the messages that start the negotiation (initiator messages), the messages that occur during the negotiation (reactor messages), and the messages that terminate the negotiation (terminator messages). See also Figure 3.8 for a detailed description of the messages.

In order for the negotiation process to start, the PA sends a *call for proposal* to the SPA so to inform it about the suggested repair by ORS and invite it to participate in a negotiation. Such messages start the negotiation process and for this reason belongs to the category of the *initiator messages*. A *call for proposal* can only be sent by the agent that plays the role of the *initiator* within the negotiation, and the time restriction of sending this message is that it can only be sent right after the initiator has obtained the suggested repair by ORS. For this reason the *call for proposal* can only occur in the initial negotiation phase, *start*, and moves the negotiation to the phase *call sent*.

The reactor messages are the intermediate messages that occur during the negotiation which do not start the negotiation nor terminate it. These messages can be sent by the initiator or by the participant, depending on the phase of the negotiation. Such messages are: *receive call*, *refuse call*, *propose*, *accept proposal* and *refuse proposal*. The *receive call* message can only be sent by the participant, only after the initiator has sent the *call for proposal* in order to inform that it has received it. The negotiation phase that corresponds to this message is the *call sent* phase. Right after the *call for proposal* has been received, the agents act according to their utility value. If the agent participant wants to participate in the negotiation, it sends a *propose* to the initiator, and the next negotiation phase will be *proposal sent*. Otherwise, the agent participant refuses to participate in the negotiation and sends a *refuse call* which will lead the negotiation to the phase *call refused*. After the participant's refusal, it sends a *failure* message to indicate that it is not willing to negotiate and leaves the negotiation. Then, the phase that the process is moved to the *participant left the negotiation*.

If a proposal has been sent by the participant, the initiator receives the proposal and decides whether it will accept it or not. If the initiator accepts the proposal, it sends

back to the SPA a *proposal accepted* message to inform that it accepts the proposal, and the negotiation is then moved to the *proposal accepted* phase. If the initiator decides to reject the proposal, it sends the SPA a *refuse proposal* message and the negotiation is transited in the phase *proposal refused*.

The negotiation terminates when the phase *end* is reached with the messages *initiator does the repair* or *spa does the repair* which indicate the outcome of this negotiation round. If the SPA has refused to participate in the negotiation, and the current negotiation phase is *participant left the negotiation*, the outcome can either be that the PA will perform the repair or that the repair will not be performed at all. So the corresponding message *initiator does the repair* is then sent, and the negotiation transits in the phase *end*. The *initiator does the repair* is also sent by the PA if it has refused the proposal from the SPA and the current phase is *proposal refused*. If the participant's proposal is accepted by the PA, and the current negotiation phase is *proposal accepted*, the SPA sends a *participant does the repair* to the PA and the negotiation process moves to the phase *end*, where it terminates.

### 3.3.2 Negotiation rules and obligations

In order to ensure that no deadlocks occur during the negotiation process, that no agent's actions conflict, and to guarantee that an agreement is reached in all possible situations, we define within our protocol the rules that govern the process. These rules specify the agent interactions, the negotiation stages, what constitutes a legal agreement and the obligations the agents have. For all these reasons we define three kinds of rules: *the negotiation rules*, *the agreement rules* and *the obligation rules*.

The *negotiation rules* are the rules that define the legal actions of the agents during the negotiation. In more detail, they restrict the set of the applicable messages depending on the current negotiation phase. In this way the agents cannot exchange messages in a random way, but instead they follow a common reasoning that guarantees at any case that an agreement is reached. The agreement rules are of the form *negotiationRule(Action, Negotiation Object, Negotiation Party, Preconditions, Effects)* where *Action* is the action that this rule is about, e.g. *Message*; *NegotiationObject* is the repair over which agents are negotiating; *Negotiation Party* is the role of the agent that is permitted to perform the *Action*, i.e. *initiator* or *participant*; *Preconditions* is a list of all the preconditions that must be true in order the *Action* to be performed, i.e. the current negotiation phase; *Effects* are the effects of this *Action*, i.e. the new

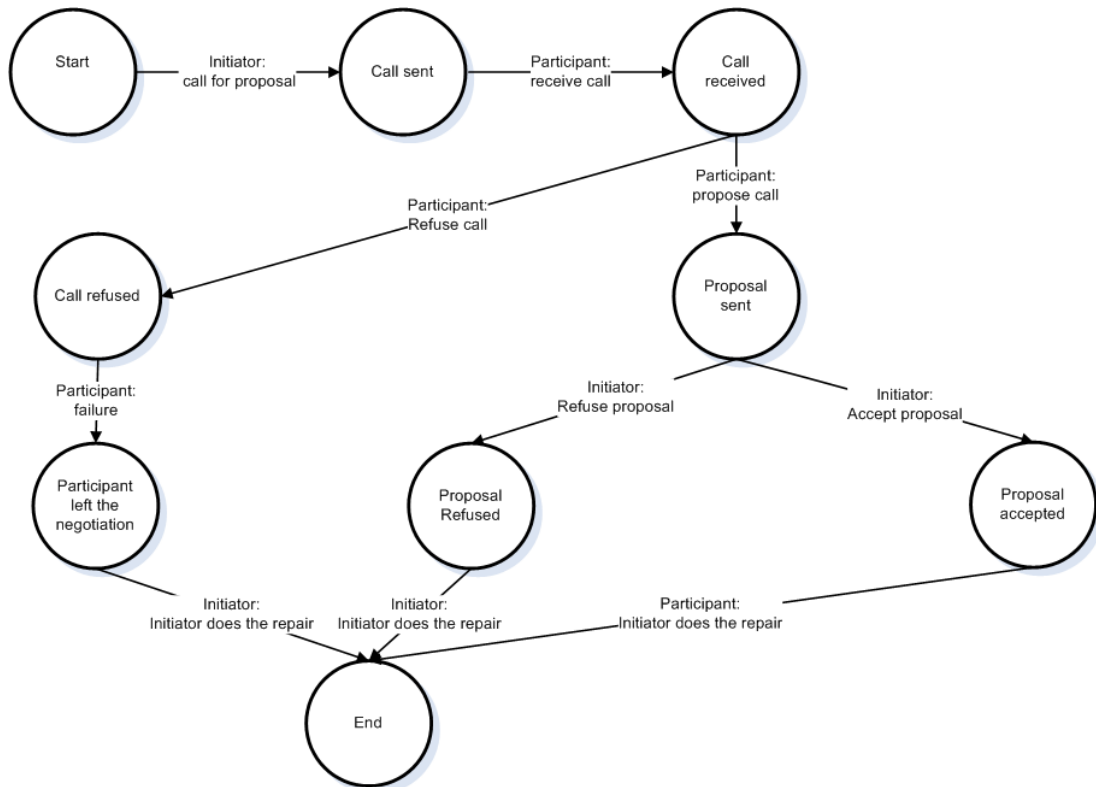


Figure 3.8: The negotiation phases and the corresponding messages

negotiation phase.

The agreement rules constitute the legal agreement and when this agreement is achieved. An example, an agreement rule states that if a proposal was sent by an agent  $i$  and this proposal was accepted by agent  $j$  then the agreement is that agent  $i$  is allocated with the repair. Lack of these rules would mean that we allow the agents to make assumptions about the agreement. For instance, let us consider the situation described above: agent  $i$  sends a proposal for the repair to agent  $j$  but the later agent instead of accepting it, it rejects it. Based on the negotiation protocol, in this case the agreement is that agent  $j$  will perform the repair. But in case of no agreement rules, agent  $i$  can be easily misled to the assumption that the proposal it sent entailed an agreement of performing the repair.

Finally we specify the obligation rules, which are concerned with the obligations of the agreed agent to perform the repair. Once an agreement is reached about which agent will perform the repair, the agreed agent is obliged to perform it and it has no right to change this agreement. At this point we should clarify that we believe that the participants are honest: an agent makes a proposal only if it can perform the repair and if it is allocated with the repair will always fulfill this agreement.

Agreement and obligation rules are of the form  
*rule name(Action, NegotiationObject, Negotiation Party, Negotiation Phase, Preconditions, Effect)*

where *rule Name* is the name of each rule, i.e. *Agreement Rule Action* is the action that has as a result an agreement to be reached, *Negotiation Object* is the repair that the agreement is about; *Negotiation Party* is the set of the all the agents this rule concerns with; *Negotiation Phase* is the negotiation stage that the agreement occurs; *Precondition* is the set of preconditions which must be true in order for the agreement to be reached, i.e. the agents' utilities; *Effect* is the effect of this agreement.

### 3.3.2.1 Preconditions and effects of the exchanged messages

To ensure that the messages are not exchanged in an arbitrary way and to restrict the set of permitted messages each negotiation participant can send, we strictly define the messages' preconditions and effects. In this way the messages occurrence is controlled with respect to the negotiation phases we have defined and to the agents' utility. The rules we define concerning the messages are :

- The message *call for proposal* can only be sent during the negotiation phase *start*

and only by the agent assigned with the role *initiator*. The effect of this message is that the next negotiation phase is the *call sent*.

- Only the message *call received* can be sent during the negotiation phase *call sent* and only by the the agent assigned with the role *participant*. The effect of this message is that the next negotiation phase is *call received*.
- The message *propose* can only be sent during the negotiation phase *call received* if and only if the agent that sends it is assigned with the role *participant* and  $utility_{participant} > 0$ . The effect of this message is that the next negotiation phase is *proposal sent*.
- The message *refuse call* can only be sent during the negotiation phase *call received* if and only if the agent that sends it is assigned with the role *participant* and  $utility_{participant} = 0$ . The effect of this message is that the next negotiation phase is *call refused*.
- The message *accept proposal* can only be sent during the negotiation phase *proposal sent* if and only if the agent that sends it is assigned with the role *initiator* and  $utility_{participant} > utility_{initiator}$ . The effect of this message is that the next negotiation phase is *proposal accepted*.
- The message *refuse proposal* can only be sent during the negotiation phase *proposal sent* if and only if the agent that sends it is assigned with the role *initiator* and  $utility_{participant} < utility_{initiator}$ . The effect of this message is that the next negotiation phase is *proposal refused*.
- The message *failure* can only be sent during the negotiation phase *call refused* if and only if the agent that sends it is assigned with the role *participant* and  $utility_{participant} = 0$ . The effect of this message is that the next negotiation phase is *participant left the negotiation*.
- The message *initiator does the repair* can only be sent during the negotiation phases *proposal refused* and *participant left the negotiation* if and only if the agent that sends it has the role *initiator* and  $utility_{participant} < utility_{initiator}$ . The effect of this message is that the next negotiation phase is *end*.
- The message *participant does the repair* can only be sent during the negotiation phase *proposal accepted* if and only if the agent that sends it is assigned with the

role *participant* and  $utility_{participant} > utility_{initiator}$ . The effect of this message is that the next negotiation phase is *end*.

Moreover we add the restrictions:

- An agent is assigned with the role of *initiator* if and only if this agent is also the PA.
- An agent is assigned with the role of *participant* if and only if this agent is also a SPA.
- The negotiation process starts only when the suggested repair is obtained from ORS and when the PA informs the SPA about the repair.
- No offer can be made nor an agreement can be reached if not both agents have formed their utility suggested repair.

### 3.3.2.2 Obligations

In addition to the negotiation rules, we define a simple obligation rule that concerns the agent which is assigned with the repair. This obligation refers to what the agents have agreed to, and declares that once an agreement has been reached this agreement cannot be undone, and the agent that is allocated with the repair is obliged to perform it.

## 3.4 Summary

In this chapter we described the *one to one* negotiation protocol we have defined for ORS purposes. The negotiation domain is defined in terms of the agents that participate in the negotiation, the messages that the agents exchange, the negotiation object and the cost function.

The agent set is constituted by *three* types of agents, including the expert agent as well, while only *two* of them have an active role: the initiator which starts the negotiation and the participant. The negotiation object set decomposes to *seven* repairs and includes the essential information of each repair. The set of permitted messages that the agents can exchange during the negotiation includes *nine* messages. Each of the messages corresponds to a specific negotiation phase, and is described by the corresponding negotiation rules.

The initiator starts the negotiation by sending a *call for proposal* request to the participant. The participant in its turn, based on its utility for the suggested repair, can either choose to participate in the negotiation by making an offer or refuse to participate if the cost of performing the repair is too high. The initiator will accept the participant's proposal if the social welfare of this agreement is the maximal; otherwise it will reject the proposal and will perform the repair. If an agreement is not reached, i.e. the participant chooses not to participate in the negotiation, then the repair is allocated to the initiator under the condition that the initiator's cost of performing this repair is not the highest possible.

Beyond the negotiation rules, we have also introduced the existence of an *expert agent* which does not participate actively in the negotiation. Instead this agent is utilised by the other agents, in cases where an agreement is not possible to be reached, in order to make a suggestion about which agent should perform the repair. The existence of this expert agent is justified by the fact that it is useful for the system an agreement to be reached, so that the initial interaction between the initiator (PA) and the participant (SPA) to be completed with success. At this point we should clarify that the agents are not obliged to obey this *expert's* suggestion, but instead we provide them with the freedom to decide whether or not they will respect it and will act accordingly. The implementation of how the agents find and choose the expert agent is currently very simple. However the design is extensible to more sophisticated and complicated mechanisms. expert agent is currently very simple, but much more sophisticated

The design of this protocol guarantees that the process will never reach a deadlock, while the rules defined are simple and easy to understand by the participating agents. Another benefit of this protocol is that the communication between the agents is minimised so as to keep the computational cost low. This is achieved by specifying a small set of exchangeable messages, which are carefully designed to cover all possible interactions. Furthermore, we have ensured that the agreement that is reached in every negotiation round maximises the social welfare.

# Chapter 4

## Negotiation Ontology

### 4.1 Introduction

It is essential for the agents that participate in a negotiation system to share a common predefined protocol, which describes the interactions and the capabilities of the agents, the rules of the negotiation, and it clarifies the type of the permitted messages the agents can exchange. By entering the negotiation, the agents, are assumed to be aware of this protocol and to agree to its rules *a priori*.

Common practice in multi-agent systems, is to hard code the agents with the negotiation protocol. Hard coding implies that each agent's code contains the protocol that corresponds to the negotiation system to which it belongs. In this way, for a successful negotiation, all the participating agents must explicitly follow the same protocol. However, as the need for open multi-agent systems grows, the hard coding techniques are limiting the agents' interactions, since they either force the agents to use one of the protocols they already know, or to be reprogrammed with a new one.

In this project, we follow a completely different approach. Instead of making the protocol explicit to the agents, we make the protocol available to the agents by creating an ontology for it. In this way we do not force the agents to be explicitly coded with our protocol, while we use interactions between agents that come from different systems. Each time an agent enters the system, it utilises the negotiation ontology to be informed about all the essential features of the negotiation.

The negotiation ontology models the negotiation domain by identifying all the features the domain decomposes to, as well as the relationships between these features. Moreover, it describes the negotiation process by modelling the negotiation messages and the negotiation phases.

## 4.2 Basic Concepts

With respect to the negotiation protocol we defined previously, we create a top-down ontology at the highest level of which we define the most general concepts that characterise the protocol: *negotiation protocol*, *negotiation party*, *negotiation object*, *negotiation message*, *negotiation phase*, *negotiation rule* and *utility* as shown in Figure 4.1.

- *Negotiation Protocol* describes the rules of encounter which during the negotiation process are followed by the negotiation participants. The permitted exchanged messages as well as the sequence of exchanging them, the conditions and the permitted offers are described.
- *Negotiation Object* describes the object that is negotiated by the agents, the suggested by ORS repair.
- *Negotiation Message* describes the permitted messages that the participating agents can exchange in order to express their preferences during the negotiation. It is categorised into : *Negotiation Message Initiator*, *Negotiation Message Reactor*, *Negotiation Message Terminator*.
- *Negotiation Phase* describes all the possible negotiation stages.
- *Negotiation Party* is the set of all the agents that can participate in the negotiation.
- *Rule* consists of the set of all the rules that govern the protocol, the agents' interactions, the messages and the specifications of the negotiation phases.

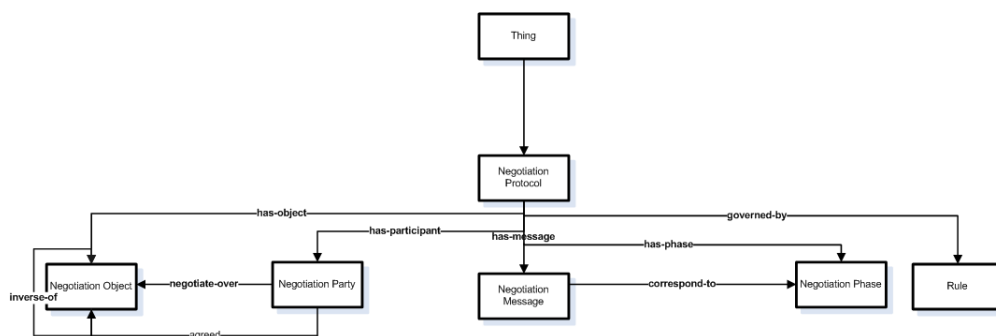


Figure 4.1: The most general concepts of the protocol

The top level concept of this hierarchy is *Thing* and the direct subclass is the class *Negotiation Protocol* which is defined as the super class since it is the most generic definition used by this ontology. In this way we provide to the ontology the flexibility of defining more than one one protocol within this ontology. These subclasses do not represent an *is-a* relation with the *Negotiation Protocol* since they are not taxonomic in nature, as [Wurman et al., 2001] states. For this reason we describe the interactions between them by defining the *ad hoc* relations  $\{ has-participant, has-object, has-phase, governed-by, has-message, has-phase \}$

All these relations have as domain the class *Negotiation Protocol* and as range the corresponding subclasses. In more detail:

- The relation *has-participant* has range the class *Negotiation Party*.
- The relation *has-object* has as range the class *Negotiation Object*.
- The relation *has-phase* has as range the class *Negotiation Phase*.
- The relation *governed-by* ranges to the class *Rule*
- The relation *has-message* ranges to the class *Negotiation Messages*
- The relation *has-phase* ranges to the class *Negotiation Phase*.
- 

All these relations share the same characteristic, they are *irreflexive*. As an example, for the *has-participant* relation  $\forall x \in NegotiationProtocol, \neg has-participant(x,x)$  holds. This means that for all the instances  $x$  belonging to the class *Neotiation Protocol*, it is not possible the negotiation protocol  $x$  to have as a participant itself.

A relation of the form *has-participant*( $x,pa$ ) says that a negotiation protocol  $x$  has  $pa$  as a participant. Furthermore we characterise this relation as *irreflexive* since  $\forall x \in NegotiationProtocol, \neg has-participant(x,x)$ .

After having defined the class hierarchy, we have to make sure that the disjointness among classes, where it is necessary, is defined as well. Considering the definition of class disjointness: two classes  $X$  and  $Y$  are disjoint if and only if an instance  $a \in X$  cannot occur in class  $Y$  at the same time. In the negotiation ontology we define the disjoint classes below:

- The class *Negotiation Message* is disjoint with the class *Negotiation Party* because

$$\forall x \in NegotiationMessage \implies x \notin NegotiationParty$$

- The class *Negotiation Message* is disjoint with the class *Negotiation Phase* because

$$\forall x \in \text{NegotiationMessage} \implies x \notin \text{NegotiationPhase}$$

- The class *Negotiation Message* is disjoint with the class *Negotiation Party* because

$$\forall x \in \text{NegotiationMessage} \implies x \notin \text{NegotiationParty}$$

- The class *Negotiation Message* is disjoint with the class *Rule* because

$$\forall x \in \text{NegotiationMessage} \implies x \notin \text{Rule}$$

- The class *Negotiation Message* is disjoint with the class *Negotiation Object* because

$$\forall x \in \text{NegotiationMessage} \implies x \notin \text{NegotiationObject}$$

- The class *Negotiation Phase* is disjoint with the class *Negotiation Party* because

$$\forall \psi \in \text{NegotiationPhase} \implies \psi \notin \text{NegotiationParty}$$

- The class *Negotiation Phase* is disjoint with the class *Rule* because

$$\forall \psi \in \text{NegotiationPhase} \implies \psi \notin \text{Rule}$$

- The class *Negotiation Phase* is disjoint with the class *Negotiation Object* because

$$\forall \psi \in \text{NegotiationPhase} \implies \psi \notin \text{NegotiationObject}$$

- The class *Negotiation Object* is disjoint with the class *Rule* because

$$\forall \omega \in \text{NegotiationObject} \implies \omega \notin \text{Rule}$$

- The class *Negotiation Object* is disjoint with the class *Negotiation Party* because

$$\forall \omega \in \text{NegotiationObject} \implies \omega \notin \text{NegotiationParty}$$

- The class *Negotiation Party* is disjoint with the class *Rule* because

$$\forall \phi \in \text{NegotiationObject} \implies \phi \notin \text{Rule}$$

## 4.3 Concept Relations and Instances

The lower level concepts specify the roles that the participating agents can have during the negotiation, the different kinds of negotiation objects that can occur, the permitted messages the agents can exchange, as well as the rules that govern the different phases of the negotiation and the interactions between the agents.

### 4.3.1 Negotiation Message

The class *Negotiation Message* refers to the set of all negotiating messages and is decomposed into three categories. Consequently, we define three categories as subclasses of the class *Negotiation Message*: the *Negotiation Message Initiator*, the *Negotiation Message Reactor* and the *Negotiation Message Terminator*. We distinguish the permitted negotiation messages into these three categories since, depending on the exchanged message, the corresponding stage of the negotiation is either the beginning of the negotiation, or one of the mediator phases, or the termination of the negotiation.

The instances that we determine for these classes are the permitted messages the agents can exchange, which differ from one another on the negotiation phase they can occur; the only instance that belongs to the *Negotiation Message Initiator* is the *call for proposal* message, which is the only message that an agent can send to start the negotiation process. The *Negotiation Message Reactor* class contains the instances *receive call*, *propose*, *refuse call*, *refuse proposal*, *failure*, *accept proposal* since these messages occur during the negotiation and they do not initialise or terminate it. The *Negotiation Message Terminator* class contains the instances: *initiator does the repair* and *participant does the repair*, since the occurrence of one of these two messages terminates the process. The reason we chose to represent the permitted messages as instances and not as classes is that they represent the most specific concepts of this category. So by defining them as instances we enable the reuse of this ontology by different protocols (see future work in Chapter 8). Defining each message as a class would restrict the generality that now the ontology offers. The applicability of each message with respect to the current negotiation stage, is defined in terms of the negotiation phase that belongs to the preconditions of each message. Based on this fact, these three classes are disjoint, since an instance belonging to one of them cannot occur in any of the other classes as well. As an example, the instance message *call for proposal* that belongs to the *Negotiation Message Initiator* has been restricted to occur only at the initial negotiation phase, so it is not possible for it to be an instance of the

*Negotiation Message Reactor* or the *Negotiation Message Terminator*.

The class hierarchy presented, which represents an is-a relation, is shown in Figure 4.2.

In addition we define the relation *correspond-to* which describes the interaction between the classes (*Negotiation Message*, *Negotiation Phase*). This relation says that every instance of the *Negotiation Message* class corresponds to an instance of the *Negotiation Phase* class. In addition we characterise this relation as irreflexive because  $\forall x \in \text{NegotiationMessage}, \neg \text{correspond-to}(x,x)$ .

### 4.3.2 Negotiation Party

The set of the agents that can occur in the negotiation domain  $A_g$  is constituted by the *initiator*, *participant*, and *expert* agents. However, the *expert* is not considered as an active agent during the negotiation, thus the class *Negotiation Party* represents the set of all the agents that are permitted to participate in the negotiation: the *initiator* and the *participant*.

For this reason we define as a subclass of the class *Negotiation Party*, the *Negotiation Party Role* class and the corresponding instances of this class, *Initiator* and *Participant*. The *Initiator* instance corresponds to the agents that start the negotiation, while the *Participant* instance corresponds to the agents that participate in the negotiation by accepting a request sent by an *Initiator* agent. We chose to define the roles the agents can have during the negotiation process as instances, since they are the most specific concepts that can occur in this category.

In addition we define a relationship between the instances of these classes, the *has-role* binary relation as shown in Figure 4.3. This relation has the domain the *Negotiation Party* class, and ranges over the *Negotiation Party Role* class. The *has-role* relation describes the role an agent has during the negotiation. The domain of this

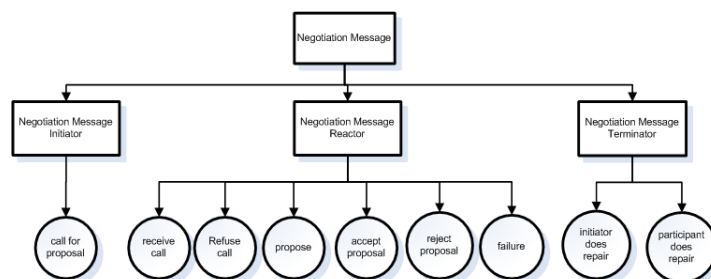


Figure 4.2: The Negotiation Message class hierarchy

relation is the class *Negotiation Party* and the range is the class *Negotiation Party Role*. For instance, the relation *PA has-role initiator* means that the PA is assigned with the role of the *initiator* during the negotiation, while we can infer from the domain and range rules that  $PA \in \text{Negotiation Party}$ , and that  $initiator \in \text{Negotiation Party Role}$ .

The relation *has-role* is characterised by some further properties:

- It is *functional*: for a given individual this property can only hold a single value. As an example the PA can only be the initiator during a negotiation round while the SPA can only be the participant. More formally:

$$\forall x \in \text{Negotiation Party} \exists (\psi, z) \in \text{Negotiation Party Role}, \text{has-role}(x, \psi) \wedge \text{has-role}(x, z) \rightarrow \psi = z.$$

- It is *irreflexive*: none of the elements defined by this relation are related to themselves. More formally,  $\forall x \in \text{Negotiation Party} \neg \text{has-role}(x, x)$ .

Because the negotiation protocol mainly concerns interactions between the instances belonging to the classes *Negotiation Party* and *Negotiation Object*, we consider it essential to define this interaction within the ontology. For this purpose, we define the relations *negotiate-over*, *agreed* and *obligated* which have as domain the class *Negotiation Party Role* and range over the class *Negotiation Object*. In addition with the properties of each relation. First of all, all these three relations are *functional*,  $\forall x \in \text{Negotiation Party} \exists (\psi, z) \in \text{Negotiation Object} \text{negotiate-over}(x, \psi) \wedge \text{negotiate-over}(x, z) \rightarrow \psi = z$ , which is justified by the fact that for every negotiation round the object over which the agents negotiate, is exactly one. Secondly, all three relations are irreflexive. For instance,  $\forall x \in \text{Negotiation Party}, \neg \text{negotiate-over}(x, x)$ . Finally,

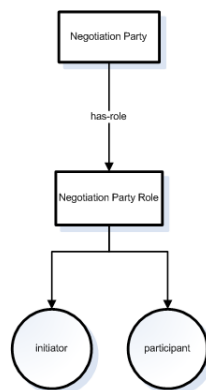


Figure 4.3: The Negotiation Party class hierarchy

the relation *agreed* is symmetric since  $\forall x \in \text{Negotiation Party}, \forall y \in \text{Negotiation Object}, \text{agreed}(x,y) \iff \text{agreed}(y,x)$

Furthermore we define the relation *send* to describe the connection between the classes *Negotiation Party Role*, which is the domain of this relation, and the *Negotiation Messages* class, which is the range of this relation. *Send* is an irreflexive relation.

### 4.3.3 Negotiation Object

The negotiation object represents the matter over which the negotiation parties negotiate. Since the *Negotiation Object* set, which this protocol deals with, is concerned with the suggested repairs and their inverses, the classification of the *Negotiation Object* is based on the different kinds of repairs that can be performed. The main categories that the repairs are involved in are the *Change Predicate* and the *Change Action Rule* categories. These two categories constitute siblings of the parent class *Negotiation Object* (with an *is-a*) and can be considered as the two main generalisations of any possible repair that can occur within the negotiation process. The *Negotiation Object* is presented in Figure 4.4.

The *Change Predicate* category deals with repairs that have to do with the predicate's name, the predicate's arity and the predicate's argument type. Consequently, we define these three repair types as siblings of the *Change Predicate* class. Finally for each of these subclasses we restrict their instances depending on ORS's repair; *predicate abstraction* and *predicate refinement* are instances of the *Change Predicate Name* class, *propositional abstraction* and *propositional refinement* are instances of the *Change Predicate Arity* class, *switch arguments*, *domain abstraction* and *domain refinement* are instances of the *Change Predicate Argument* class.

The classes that belong to the hierarchy we presented in this section are disjoint: no repair can belong to more than one repair category at the same time.

### 4.3.4 Negotiation Phase

The *Negotiation Phase* class represents the set of all the stages the negotiation process can go through. The classification we follow for the *Negotiation Phase* is similar to the one followed for the *Negotiation Message*: we categorise the negotiation phases into the initial phase, the median phases, and the final phase of the process. Consequently we define, using an *is-a* relationship, three sibling subclasses of the class *Negotiation Phase*: *Negotiation Phase Initial*, *Negotiation Phase Intermediate*, *Negotiation Phase*

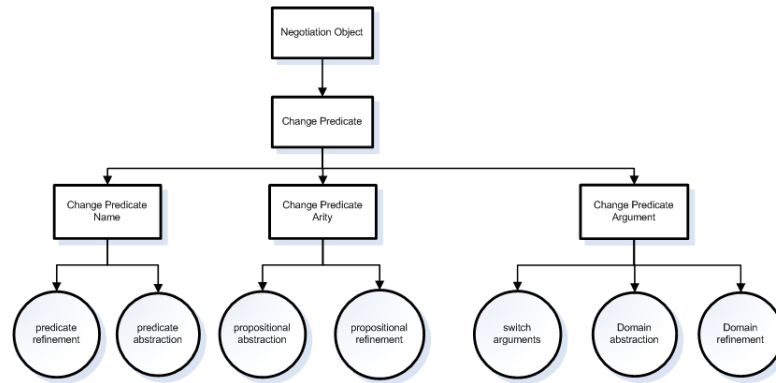


Figure 4.4: The Negotiation Object class hierarchy

*Final*. All these three subclasses are disjoint: every possible negotiation phase is carefully restricted to have a specific impact on the negotiation process. The class hierarchy is shown in Figure 4.5.

Furthermore, we restrict the instances of each subclass by defining *start* as an instance of the *Negotiation Phase Initial*: *call sent*, *call received*, *proposal sent*, *proposal refused*, *proposal accepted*, *participant left the negotiation* instances occur in the *Negotiation Phase Intermediate* class; and lastly the *end* instance occurs in the *Negotiation Phase Final* class.

Finally, since the suggested repair for the *initiator* agent is always the inverse repair suggested for the *participant*, we define the relation *inverse-of* which has domain and range the class *Negotiation Message*. This relation is *non reflexive*, *symmetric* and *functional*. First of all, it is symmetric with respect to the symmetric definition  $\forall x, y \in \text{NegotiationMessage}, \text{inverse-of}(x, y) \iff \text{inverse-of}(y, x)$ . For instance the *precondition abstraction* is the inverse repair of the *precondition refinement* if and only if the repair *precondition refinement* is the inverse of the *precondition abstraction*.

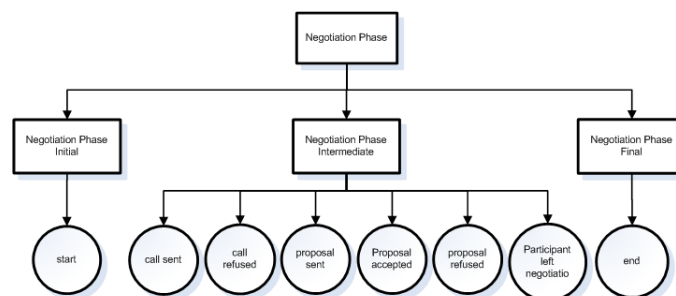


Figure 4.5: The Negotiation Phase class hierarchy

This relation is considered as *partially reflexive* and not *entirely reflexive*. As an example the *inverse-of*(*switch arguments, switch arguments*) holds, but the *inverse-of*(*predicate abstraction, predicate abstraction*) does not. So this property depends entirely on the repair.

### 4.3.5 Rules

With respect to the rules that have been specified within the negotiation protocol, the rules we consider in this ontology are the rules that deal with negotiation, agreement and the agents' obligations. Consequently we have defined as subclasses of *Rule* the sibling and disjoint class classes *Negotiation Rule*, *Agreement Rule* and *Obligation Rule*. The class hierarchy is shown in Figure 4.6.

However this classification exists to ensure the formality of the ontology, while each individual rule is defined as an axiom within the ontology. A general form of the rules described within the ontology is *rule name (Action, Agent, Preconditions, Effects)* where *Preconditions* is a conjunction of relations that must be true in order for the *Action* to be applicable to the current state and *Effects* is a conjunction of relations that describe the situation after this rule is applied.

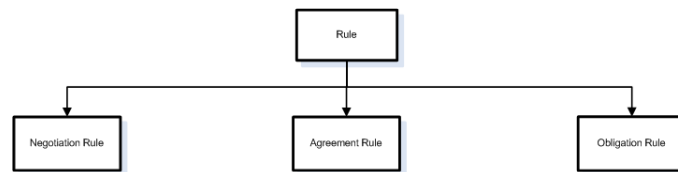


Figure 4.6: The Rule class hierarchy

### 4.3.6 Summary

In this chapter we described the design for the implementation of the negotiation ontology. The design of this ontology captures both the features of the protocol and the negotiation process itself, so in this way it provides to the agents all the essential knowledge to reason about the negotiation process.

The ontology follows a top-down decomposition by defining as the top level concept the class *Thing* and its direct subclass *Negotiation Protocol*. While both *Item* and *Negotiation Protocol* are very generic concepts, we decided to create this hierarchy because it provides the ontology with the flexibility of extending it by adding more

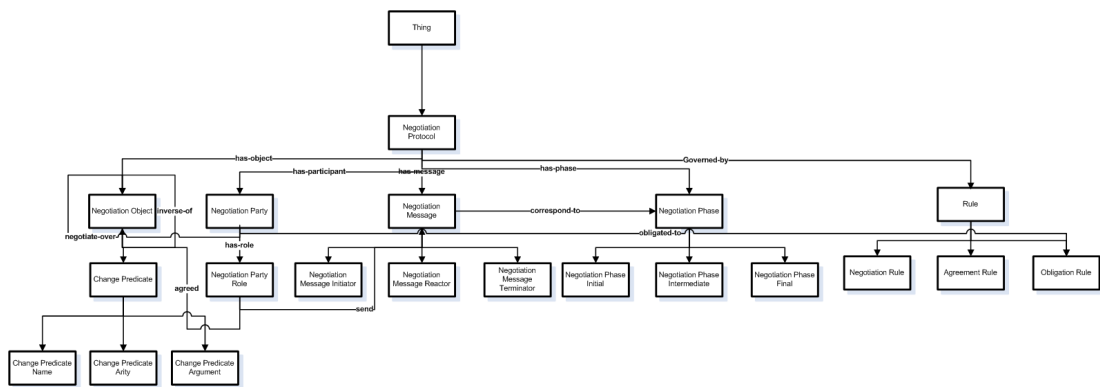


Figure 4.7: The taxonomy of the ontology

than one protocol. Instead of representing the class hierarchy by an *is-a* relationship, because the concepts do not follow a taxonomic order in the nature, we define *ad hoc* relations to represent the interactions between the classes and the instances belonging to them.

The taxonomy of this ontology is shown in Figure 4.7

# Chapter 5

## Implementation

### 5.1 Overview of the system

One of the goals of this project is to enable agent negotiation while avoiding any protocol-related hard-coding. In addition, our system has to smoothly integrate and cooperate with the existing implementation of ORS. For this reason we chose to implement our system in *Sicstus Prolog* [Sicstus, 2005] as this is implementation language of the original ORS.

Since one of our aims is not to hard-code the agents and to avoid, or at least minimise, any changes in the agents' architectures, we introduce to the system an external *service* which is utilised by the agents for negotiation-related issues. This is a negotiation specific service which receives negotiation requests depending on the agents needs. In addition this negotiation service is responsible for contacting the expert agent, in case the negotiating agents ask for it: it is responsible for tracking and informing the expert agent about the negotiation participant's request. In our system, this service is named after the purposes it fulfills: *negotiation service*. The negotiation service is provided with the negotiation ontology and is responsible for making all the essential information for successful negotiation, available to the negotiation parties.

When a communication failure occurs between the PA and the SPA, the PA obtains the suggested repair from ORS and checks its ontology for any kind of protection. After it has formed its utility, based on the protection levels of the ontological part this repair concerns with, it informs the SPA about the suggested repair. When the SPA receives the repair, it checks its ontology for any kind of protection and forms accordingly its utility for this repair. Afterwards, it informs the PA, about this utility. After this part of the communication is over, the PA invokes the negotiation service by

providing the repair and both agents utilities as shown in Figure 5.1.

The negotiation service uses the suggested repair and the agents' utilities as preconditions and having as its goal a successful negotiation round tries to form a plan based on the domain described within the negotiation ontology. A successful negotiation round is a round which has a beginning and an end, regardless of the outcome of this negotiation. The plan that is formed is constituted by the set of rules that corresponds to this negotiation round. More specifically, based on the precondition set and the negotiation ontology that contains the protocol, the negotiation service is responsible for finding the specific rules and the protocol concepts which concern that abide to these preconditions. For more details see section 5.2.3.

The negotiators (PA and SPA) can request the expert agent's opinion; it is the responsibility of the negotiation service to contact the expert agent. When the plan is formed, the negotiation service provides it to the PA and the SPA in the form of an ontology which also contains all the necessary information about the protocol, the negotiation rules and the vocabulary. After both the PA and the SPA obtain the negotiation ontology, they start negotiating as shown in Figure 5.2.

In our implementation in case the negotiators desire to consult the expert agent, the negotiation service has to be aware of and contact with the expert agent. However, there are many other possible implementations. As an example, the negotiators might be aware on an expert so they do not need the negotiation service as a mediator.

The expert agent can provide advice only if it is aware of the exact ontological object in which the mismatch occurred. So, when the participants inform the negotiation service about their desire to obtain the expert advice, they also provide the cause of the communication failure as well as their representations. After the negotiation service receives all this information, it provides it to the expert agent and obtains its advice. Afterwards, it replies to the agents with this opinion, as shown in Figure 5.3, and waits for the agents to decide whether they will follow it or not. Based on this decision, the negotiation service is going to form the negotiation plan.

## 5.2 The Negotiation Service

The negotiation service must always be available to the participating agents and follow their needs and preferences. The assigned tasks of this service ensure a smooth operation of the negotiation process. The main requirements of this service are to have access to the available expert agents, to have access to the negotiation ontology and to

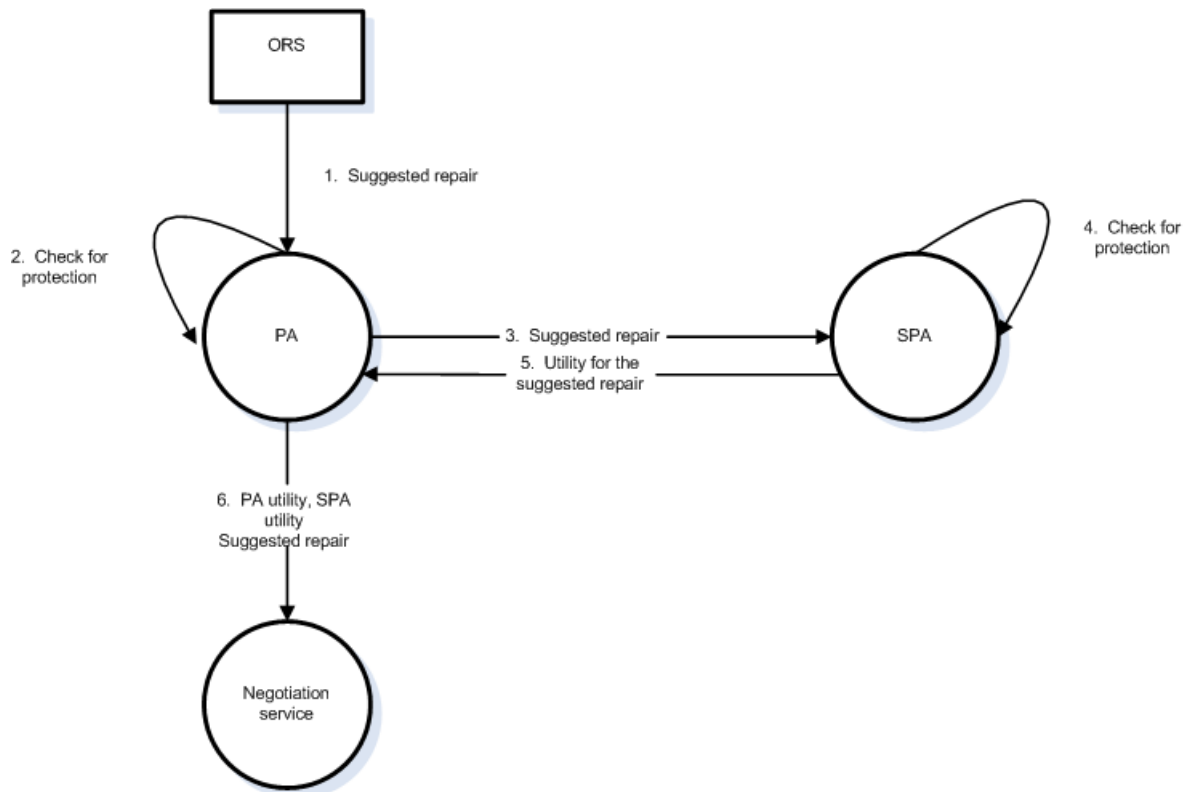


Figure 5.1: The process after a communication failure

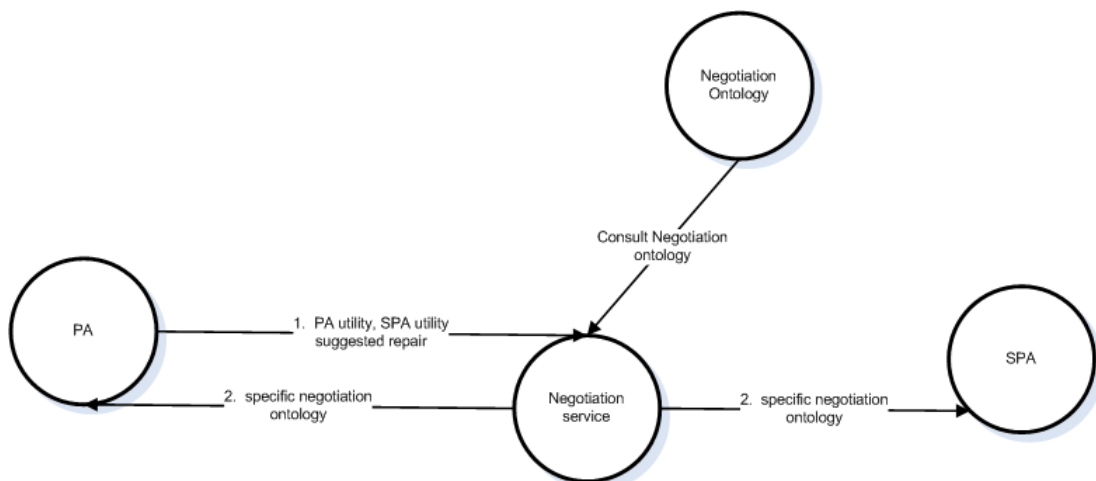


Figure 5.2: Interaction between the PA, the SPA and the negotiation service.

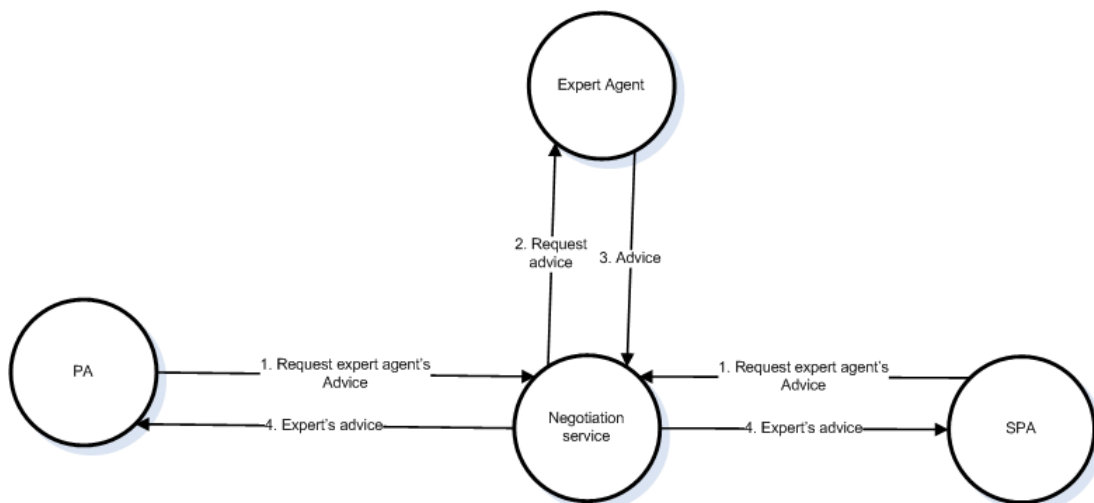


Figure 5.3: Expert Agent's Advice

be able to form the specific ontology based on the negotiators' needs. So, in order for this service to be able to serve these needs it has to be aware of the negotiation object, the participants, the participants' utilities and their ontological representations in case of an expert advice request.

The negotiation service is responsible for enabling the PA and the SPA to negotiate. This process is assured by the following specifications:

- Has access to the negotiation ontology.** In order for the negotiation service to serve the purposes of the negotiation between the PA and the SPA it must have access to the negotiation ontology. We do not hard code the protocol in this service, but instead as the PA and the SPA, has access to it. This ontology can be replaced at any time by another one that contains a different protocol. As an example the agents could suggest another negotiation ontology.
- Be aware of the repair suggested by ORS.** The negotiation service must be aware of the suggested repair in order to check if this repair can be negotiated. This is achieved by searching through the negotiation object set that is defined within the negotiation ontology. If the repair belongs to this set, then the inversed one is searched by checking for an instance related to the negotiation object with the relation *inverse-of*. This is done because the suggested repair by ORS corresponds to the PA's representation, so the suitable repair for the SPA is inverse one.

This procedure serves two purposes: first of all it is a means to ensure that the agents are able to negotiate over this repair. If a repair is not defined in the negotiation object set, this implies that it does not have the essential characteristics to be performed

by both the PA and the SPA. For instance, consider the repair *negated precondition*, which does not belong to the negotiation object set. If the negotiation service does not check the negotiation object set, it will lead the agents to negotiate over a task which by its nature can only be performed by the one of them. Consequently, there is risk of reaching an agreement for an impossible-to-perform-repair, which leads to obligation and agreement rule violation, to a waste of computation and, most importantly, to plan failure.

Secondly, the negotiation service must form the negotiation object set for this negotiation round. The negotiation object set is constituted by the initiator's negotiation object and by the participant's negotiation object: the initiator's negotiation object is the suggested by ORS repair, while the participant's is the inversed one.

- **Assigns roles to the negotiation participants.** It is the responsibility of the negotiation service to assign the PA with the role of the initiator and the SPA with the role of the participant. It takes as input these two agents' names and creates the relations *has-role(PA-Name, initiator)* and *has-role(SPA-Name, participant)* respectively.
- **Finds the relation between the participant's utilities.** The negotiation service takes as input the PA's and the SPA's utilities and assigns them as the initiator's utility and the participant's utility. It creates the relations *has-utility(initiator, Initiator-Utility)* and *has-utility(participant, Participant-Utility)*. It proceeds to the rest of its assigned operations, if and only if both participant utilities are obtained. This is achieved by checking the number of the participants for every negotiation round and comparing it to the number of the utilities it has.
- **Has access to the available expert agents.** The negotiation service must be aware of the available expert agents and have access to them, so in case of a request for expert advice it must be able to eventually reply back with this advice. Without loss of generality, in our implementation, we are assuming the expert replies immediately.
- **Be aware of whether the participants want to consult the expert agent or not.** The negotiation service has to follow the PA's and the SPA's needs and not to act based on its own beliefs; this service is not authorised to get an advice from the expert agent, unless both the participants ask for it.
- **Considers whether the negotiation participants want to follow the expert's advice.** After consulting the expert agent about the ontological mismatch, this advice cannot be acted on unless both agents agree to it. In case the participants decide to follow the expert's advice, the negotiation service has to alter the utility of the agent which will make the repair by incrementing it by *one*.

- **Forms the utility invariant for each negotiation round.** After the decision of whether the expert agent is going to be consulted and whether this advice is going to be followed, the negotiation service forms the final values of the participants and compares them. This comparison forms the invariant for this negotiation round. Finally it checks both utility values for equality to zero.
- **Forms the negotiation plan.** The participants' utilities and the negotiation object constitute the precondition set. In combination with the actions (the negotiation messages) defined within the negotiation ontology, they define the planning domain. The negotiation service has to form a plan based on these preconditions, having as a goal the creation of a complete negotiation round.
- **Distributes the plan to the agents.** Creates an ontology which contains the plan and the domain, and makes it available to the negotiation participants. This ontology provides the agents with all the essential information to reason about the protocol and to begin the negotiation process using the rules and the actions included in it.

For every negotiation round, the negotiation service repeats all these operations, as shown in Figure 5.4, according to the negotiation needs.

### 5.2.1 Consulting the expert agent

The main criterion of implementing the way the expert agent involves in the negotiation process is to ensure that the autonomy of the agents is not violated. By no means can we force the agents to ask for advice or to make this advice explicitly influence the decisions the agents make regarding the negotiation. For all these reasons we consider the expert agent's advice as an extra feature and we do not base the negotiation process on it. The participating agents are provided with all the freedom to choose whether or not they want to consult the expert.

The only case in which the expert agent can be asked for advice is if both participants agree to do so. If one of them does not desire advice, the negotiation service does not proceed to any further action. More formally, when the negotiation service is asked for the expert agent's advice, the negotiation service checks whether both participants agree on the expert agent's consultation, and, if so, it interacts with the expert agent.

The participating agents are permitted to ask for the expert agent's advice only in two situations: before the beginning of the negotiation and in the case of inability to reach an agreement, e.g. if both agents want to perform the repair. When the participating agents contact the negotiation service and provide it with their utilities, they also

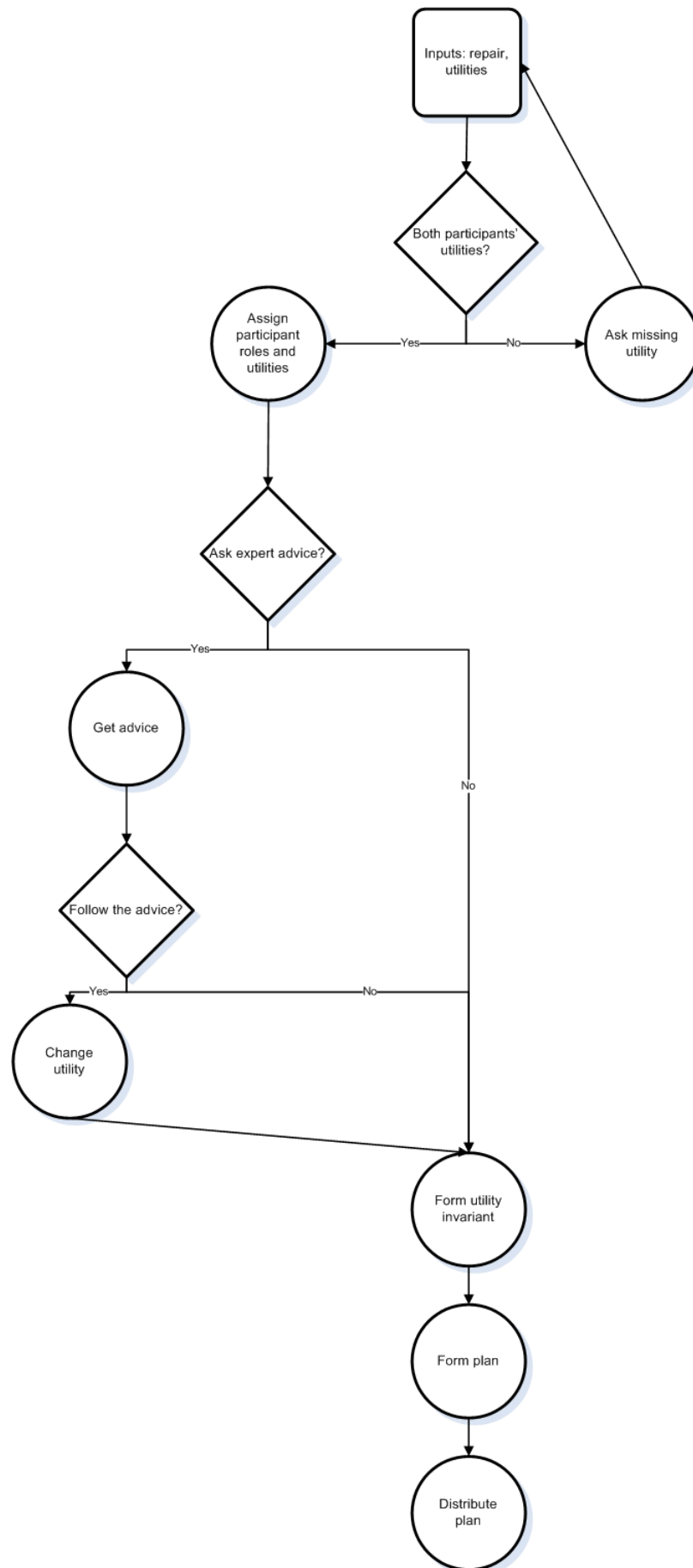


Figure 5.4: The flow of the processes assigned to the negotiation service.

inform it whether or not they want to consult the expert agent. In the case that they both agree to consult the expert, they provide the exact cause of the communication failure to the negotiation service. This information contains the ontological representations of the PA and the SPA that do not match. Then, the negotiation service consults the expert agent about the representation it thinks is the correct one. After obtaining the expert's advice, it finds which one of the participating agents has the same representation as the one the expert suggested, and returns the outcome to the participants. This outcome is the suggested agent for performing the repair, since it is this agent's representation that does not match with the one suggested by the expert. After the agents receive the outcome, they decide whether they want to follow the suggestion or not. In case they both agree to follow it, the suggested agent increments its utility by *one*.

As an example let us illustrate a communication failure from one of the existing scenarios, implemented for ORS. We are facing a communication failure between the the PA and *acceptEuropeanUnionAgent*. The PA requests from the *acceptEuropeanUnionAgent* to perform the action *acceptEuropeanUnion(greece)*. Their communication failed because of a difference on the representation of one of their preconditions. The PA's representation is *located(greece, europe)* while *acceptEuropeanUnionAgent*'s representation is *located(europe, greece)*. When the PA obtains the diagnosis and the suggested repair *switch arguments*, it informs *acceptEuropeanUnionAgent* about the repair so it can check its ontology for protection, and to form its utility. Afterwards, it sends a request to *acceptEuropeanUnionAgent*, asking whether it wants to consult the expert agent.

PA: ORS proposed the following repair: [located(greece, europe), [switch arguments, located]]. The inverse repair which I am going to negotiate is switch arguments. I am informing the SPA: *acceptEuropeanUnion* about the repair ORS suggested, so he can check. its utility.

AcceptEuropeanUnionAgent: I was asked to repair my ontology. The repair is: switch arguments. I must check if the repair can be performed. For the repair switch arguments my utility is 2. I sent my utility for this repair.

PA: I received the *acceptEuropeanUnionAgent* utility, which is 2. I want to consult an expert agent. I just asked agent *acceptEuropeanUnionAgent* if it also wants to get the expert agent advice.

AcceptEuropeanUnionAgent: \* I was asked if I want an expert advice and I replied yes.

PA: *acceptEuropeanUnionAgent* replied yes . The expert agent says that the correct one is the [located(greece,europe)] the SPA believes that the correct one is the [located(europe,greece)]. This is wrong. Therefore, the expert agent believes that the participant should make the repair. I just Informed *acceptEuropeanUnionAgent* about the advice outcome and asked it whether it wants to follow this advice.

AcceptEuropeanUnionAgent: PA informed me that the expert suggested the SPA agent. I want to follow this advice.

PA: We agreed to follow this advice.

In this example, we illustrate the dialogue between the PA and the SPA concerning the expert's advice. The PA initially wants to consult the expert agent, so it checks whether the SPA is willing as well. Since it gets a positive response it sends all the essential information about the ontological mismatch to the negotiation service, and obtains the expert's advice. Afterwards it informs the SPA about this advice, by sending which agent was suggested to perform the repair, and waits for a reply. This reply is whether the SPA wants to follow this advice.

The protocol that is followed for the expert's advice is entirely based on the agents' autonomy to decide whether they need an advice or not. We cannot force the agents to consult and/or follow the advice, or make any assumptions about their willingness to do so. Even if this means that more messages are exchanged, it is the only way to ensure that the agents proceed based on their own desires, and that any action that is taken respects both agents. For this reason, both the consultation of the expert and the change of the suggested agent's utility are the outcome of mutual agreement.

In conclusion, any action taken by the negotiation service concerning the expert agent's advice, comes after a mutual agreement between the participants. For this reason, none of the participants has the authority to ask the negotiation service to perform an action concerning the expert agent, unless the other agent has agreed to. The flow of the procedure described in this section is shown in Figure 5.5

### 5.2.2 Negotiation Ontologies

During the execution of the operations that the negotiation service is assigned with, the ontologies that this service deals with are two: the basic ontology which defines the negotiation protocol and a dynamically created one. The basic ontology contains all the rules, the vocabulary, and the features the protocol decomposes to, so to provide a fully detailed description of the protocol and the negotiation domain. This ontology is available to the negotiation service as well as to the participants.

The dynamic ontology is created *on the fly* by the service and corresponds to the specific needs for every negotiation round. This ontology can be considered as a subset of the basic negotiation ontology. It contains the concepts which characterise the negotiation protocol to as well as the description of the negotiation domain specifically for the negotiation round. For instance, the negotiation object has the form of the suggested repair, the set of the participants is constituted by the PA's and the SPA's names which have also been assigned with their negotiation roles. Not all the messages, the phases and the rules (negotiation, obligation and agreement) are included, but only these ones that concern with this round. These preferences are expressed by the participants utilities and their willingness to consult and follow the expert agent's advice.

After the creation of the dynamic ontology, the negotiation service distributes it to the participants. By obtaining the dynamic ontology, the participants have all the essential knowledge that will enable successful negotiation. In the case that the partic-

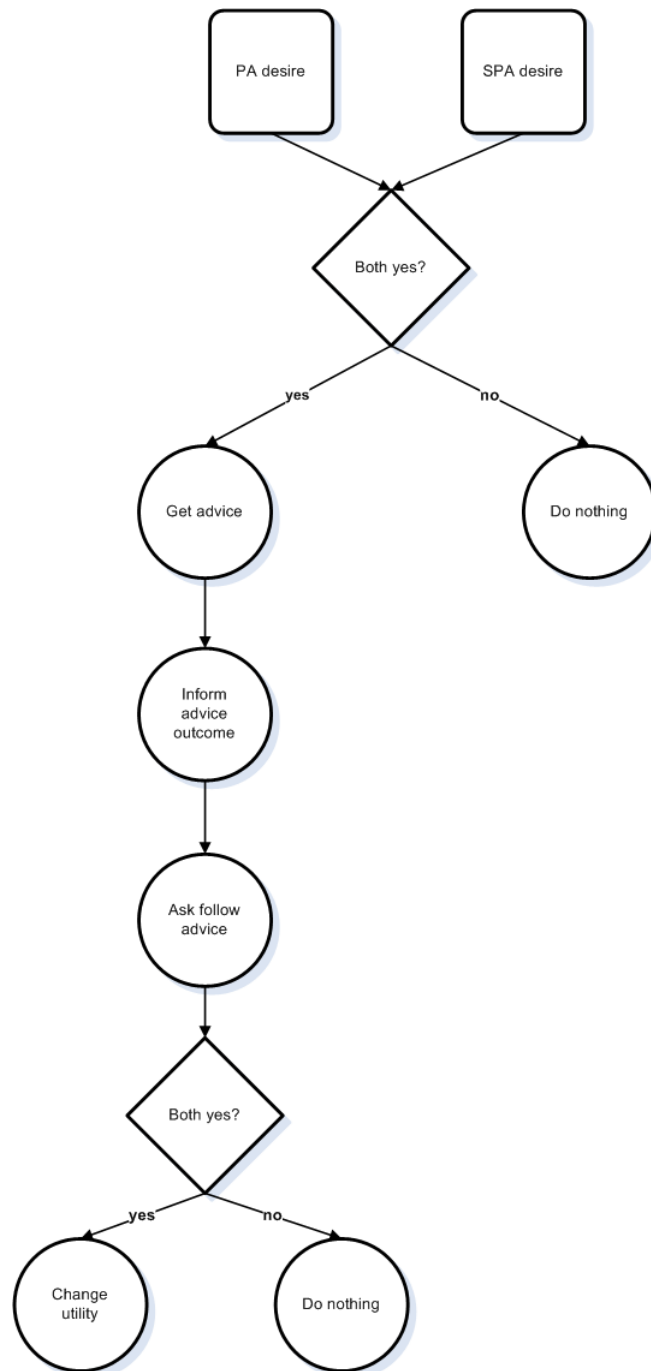


Figure 5.5: The flow of the agent expert consultation process.

Participants need more information about the negotiation protocol, they request the service to provide them with the basic ontology which contains every single aspect of the protocol and the negotiation domain.

### 5.2.2.1 Ontology Representation

Both the basic and the dynamic ontologies have to be represented in a way that is acceptable to all the agents within our system and to have a broadly recognised format. Another criterion for the implementation of the ontologies is to smoothly integrate with the agent communication system and to be usable by the negotiation service. Both negotiation ontologies deal with first order logic and reason about objects, relations and functions. Based on this need and on the requirements of the system, we considered Prolog as the best candidate language for creating the negotiation ontologies. The structure of each ontology is similar to the one followed for the PA's and the SPA's ontologies. This is done for the simple reason that it follows a KIF based format, so in case of a future need to translate the Prolog ontologies to KIF, that could be easier to achieve (see Chapter 8).

The idea we had at the beginning was to implement the ontologies using *KIF* [Genesereth, 1991]. KIF stands for *Knowledge Interchange Format* and is a language broadly used for ontology engineering. KIF is used for the representation of the agents' ontologies. In addition, these ontologies are translated into Prolog, to serve the requirements of the system. So it would be reasonable to implement the negotiation ontologies in KIF in the same way the ontologies of the agents are implemented.

Since the negotiation service is implemented in Prolog, the idea was to take advantage of the translation system offered by ORS and to bring the negotiation ontology into an understandable to the negotiation service form. After this translation the negotiation service could execute its operations. But we did not find this kind of approach because of the following reasons. First of all, for KIF based ontology, the negotiation service would have to translate it into Prolog and then proceed with its operations which deal with this ontology reasoning and creating the dynamic one. After the creation of the dynamic ontology, the negotiation service would have to translate this ontology from Prolog to KIF so to distribute it to the participants. The dynamic ontology, among the other uses it has, also serves the negotiation communication purposes of the participants. Since the agent communication is entirely based on Prolog, this would mean that the dynamic ontology would have to be translated by the agents from KIF to Prolog. In addition if the participants chose to request from the negotiation service the basic negotiation ontology, an extra translation of this ontology would be essential. In conclusion, we considered this approach as suboptimal since it needs five translations in total. So, we chose to implement the ontologies in Prolog.

The class hierarchy is represented by predicates of the form *subclass(Class, Parent Class)*, e.g. *subclass(negotiation-protocol, thing)* means *class negotiation-protocol is a subclass of class thing*. The instances are represented by the predicate *instanceOf(Instance, Class)*, e.g. *instanceOf(cfp, negotiation-message-initiator)* which translates to *cfp is an instance of the negotiationMessageInitiator class*. The class and instance relations are of the form *relation(Domain Class, Range Class)* e.g. *has-object(negotiation-protocol, negotiation-object)*, which means that the negotiation protocol has the change-predicate-name as an (negotiation) object. In addition we define each relation using the term *property(relation)*; for instance *property(has-phase( negotiation-protocol, negotiation-phase))* defines the *has-phase* as a property. Finally all the rules described by the protocol are represented as a conjunction of literals, *rule-name(Action, Preconditions, Effects)*, where the preconditions are represented as a list of the facts that have to be true for the action to be applicable, while the effects are represented as a list that contains all the facts that are true after the action is applied.

Using Prolog for representing our ontologies, we keep a balance between expressiveness and reasoning support. We create ontologies easy to be understood from our system and from the agents.

### 5.2.3 Planning System

The negotiation ontology aims to provide all the essential knowledge for a successful negotiation. So, in order to dynamically create the ontologies that are specific to each negotiation round, we provided the negotiation system with a planner. The negotiation object, the negotiators' utilities, the negotiators' roles and the basic negotiation ontology (in which the protocol is included) constitute the preconditions of this planner. Based on these facts, the planner creates the plan which is composed of the negotiation rules that correspond to these preconditions.

The idea that we first had was to provide the agents with this ontology and use it as domain knowledge; each agent could construct a negotiation plan based on its utility. Indeed, this idea was the starting point for this project and created the basis of the implementation. Our initial approach was to utilise the negotiation service only for distributing the negotiation ontology to the participants. The participants, in turn, would have to consult this ontology, interpret it, and form a plan based on each ones utility. This plan would constitute the strategy each agent would follow during the negotiation.

One limitation of this approach is that the SPAs are not provided with a planning system, so automatically this creates the need of implementing extra planners, possibly by hard-coding them. In addition, as mentioned in Chapter 3, our negotiation domain limits the set of all the possible negotiation strategies to *two*: an agent can either make a proposal for performing the repair or refuse to negotiate. Secondly, based on the defined negotiation protocol an agent which has utility greater than *zero* always makes a proposal for a repair. Finally the agreement that is reached implies that the agent with the greatest utility is the one that will perform the repair, so the social welfare will be at most one. In conclusion, each agent has only one strategy depending on its utility.

All these reasons justify the fact that the agents do not really need to form a plan for the negotiation process since the rules are straightforward; the agents only need to follow them in order to negotiate successfully. So instead of providing the agents with planners we implemented a simple planner as part of the negotiation service. This planner chooses the applicable messages for each negotiation round.

### 5.2.3.1 Planning Domain

In order to describe the main elements of our planning problem, let's consider the negotiation process as a state transition system  $\Sigma = (S, A, E, \gamma)$  where

- $S = \{s_0, s_1, s_2, \dots, s_g\}$  is the finite set of states (the negotiation phases),
- $A = \{\text{cfp, receive-call, refuse-call, propose, refuse-proposal, accept-proposal, failure, initiator-does-repair, participant-does-repair}\}$  is the finite set of actions (the negotiation messages)
- $E = \emptyset$  is the set of events, which is an empty set
- and  $\gamma$  a state transition function.

We consider our system to be static, i.e. there are no internal dynamics that would effect a state; only the application of an action changes the current state. With respect to the phases and messages defined in the negotiation protocol, the system is deterministic. For every action  $\alpha$ ,  $|\gamma(s, \alpha)| \leq 1$ . In case that an action is applicable to a state, the application of this action brings the system to a single other state. Another characteristic of the planning domain, is that the planner deals with a restricted goal. The objective is to bring the system to the state end, through a sequence of actions and states. This objective can only be achieved by a linear sequence of actions, under the assumptions that our system is deterministic, fully observable, static and finite.

In addition our model is not concerned with any time duration; actions are just instantaneous state transitions. Based on these restrictions, our system can be characterised as a restricted model in which the problem we are dealing with is of the form: *Given a system  $\Sigma = (S, A, E, \gamma)$ , the initial state  $s_0 = \text{start}$  and the goal state  $S_g = \{\text{end}\}$  find a sequence of actions  $a_1, a_2, \dots$  which correspond to a sequence of states  $s_0, s_1, \dots$  such that  $s_1 \in \gamma(s_0, a_1), s_2 \in \gamma(s_1, a_2), \dots, s_g \in \gamma(s_{g-1}, a_g)$  and  $s_g = \text{end}$ .*

We chose the classical representation for our planning domain. In classical representation, the states and the actions are a conjunction of first order logic literals. This choice is justified by the fact that we are dealing with a first order logic programming language, and that the classical representation is considered as the most popular representation for restricted transition systems.

Based on this representation, our planning problem is of the form  $\mathcal{P} = (A, s_0, g)$  where

- $A$  is the set of the actions (messages): the ground instances of the planning operators (the actions),
- $s_0$  is the initial state, and
- $g$  is the goal state.

A planning operator is defined as a triplet  $O = (\text{name}(O), \text{precond}(O), \text{effect}(O))$  where

- $\text{name}(O)$  is the name of the operator,
- $\text{precond}(O)$  are the preconditions of this operator,
- $\text{effect}(O)$  are the effects of this operator.

In our domain, the ground instances of the planning operators, the actions, are constituted by the negotiation messages. As an example, let us consider the propose message as the triple  $(\text{propose}, \text{precond}(\text{propose}), \text{effect}(\text{propose}))$  where:

- $\text{propose}$ , is the operator name,
- $\text{precond}(\text{propose})$ :  
 $\{\text{negotiate-over}(\text{initiator}, \text{negotiation-object}),$   
 $\text{negotiate-over}(\text{participant}, \text{negotiation-object}),$   
 $\text{send}(\text{participant}, \text{propose}),$

has-phase(negotiation-protocol, call-received),  
 correspond-to(propose, call-received),  
 has-Utility(initiator, initiator-utility),  
 has-utility(participant, participant-utility>0)}.

These preconditions declare that in order for this message to be applicable in the current state, both participating agents must negotiate over the same object, the message propose can only be sent by an agent assigned with the role of the participant, the current negotiation phase must be the call-received, the message propose must correspond to the phase call-received and that both participating utilities must be obtained while the participant's utility must be greater than *zero*.

- effect(propose): has-phase(negotiation-protocol, proposal-sent),  $\neg$ correspond-to(propose, proposal-sent),  $\neg$ send(participant, propose). The effects of this message are that the current negotiation phase is the proposal-sent, that the message propose does not correspond to the negotiation phase proposal-sent and that at this state the participant cannot send the message propose.

As an applicable action we define an action  $\alpha \in A$  such that  $\text{precond}(\alpha) \subseteq s$ , where  $s$  is the current state, and  $\text{effect}(\alpha) \cap s = \emptyset$ . The result of applying the action  $\alpha$  to the current state  $s$ , is given by:  $\gamma(s, \alpha) = (s - \text{effect}^-(\alpha) \cup \text{effect}^+(\alpha))$ . The  $\text{effect}^-(\alpha)$  is the set of the negative effects of action  $\alpha$  while  $\text{effect}^+(\alpha)$  are the positive ones.

The initial state  $s_0$  defined for the planning problem is formed by the initial negotiation phase, the agents' utilities and the initiator as the agent that can send a message in this round. More formally,  $s_0 = \{\text{send}(\text{initiator}, m_1), \text{has-utility}(\text{initiator}, \text{initiator-utility}), \text{has-utility}(\text{participant}, \text{participant-utility}), \text{negotiate-over}(\text{initiator}, \text{negotiation-object}), \text{negotiate-over}(\text{participant}, \text{participant-object}), \text{correspond-to}(m_1, \text{start})\}$ .

Finally the goal state  $s_g$  is the state where the following literals are satisfied  $\{\text{has-phase}(\text{negotiation-protocol}, \text{end}), \text{agreed}(\text{negotiation-party-role}, \text{negotiation-object}), \text{obligated}(\text{negotiation-party-role}, \text{negotiation-object})\}$ .

### 5.2.3.2 Planning Algorithm

The planner takes as inputs the initial state, the agents' utilities, a description of the system provided in the negotiation ontology and the goal state, and generates a plan. The goal state does not correspond to a specific outcome for the negotiation process, but just the state in which the negotiation ends. The planner does not produce a plan that guarantees an agreement, but instead it guarantees that no deadlocks occur and

that all the actions performed during the process correspond to the ones defined within the protocol. So, this plan consists of a sequence of actions having as initial state the beginning of the negotiation, and as a goal the terminal negotiation phase.

Let  $\mathcal{P} = (A, s_0, g)$  be the planning problem the planner deals with. The steps the planner follows are:

```

Data:  $\mathcal{P} = (A, s_0, g)$ 
Result: plan  $p$ 
 $s = s_0$ 
 $p \leftarrow \emptyset$ 
 $applicable \leftarrow \emptyset$ 
foreach  $\alpha \in A$  do
  if  $precond(a) \subseteq s$  then
     $applicable \leftarrow applicable \cup \{\alpha\}$ 
     $p \leftarrow p \cup \{\alpha\}$ 
     $s \leftarrow \gamma(s, \alpha)$ 
  if  $s$  satisfies  $g$  then
    return  $p$ 
if  $applicable = \emptyset$  then
  return failure

```

**Algorithm 1:** Planning algorithm

In more detail, the problem solver has initially the plan and the applicable actions set to the empty set and it begins searching for applicable actions starting with the initial state. If it finds an action which has a set of preconditions belonging to the current state, then this action is added both to the plan set and the applicable set, and the next state is formed by the function  $\gamma(s, \alpha) = (s - effect^-(\alpha)) \cup effect^+(\alpha)$ . This procedure repeats until a state which satisfies the goal state is reached. In case that the applicable set is empty, it returns a failure.

This algorithm performs state space planning, since the search space is a subset of the state space. The search approach that is followed is data driven: given some data inputs, a set of rules and the set of legal actions (the defined messages), the algorithm searches towards the goal expanding the search space. Every state that is produced is closer to the goal state. This procedure continues until the goal state is reached. The planning approach followed in our system, is sound: the plan it returns as a solution, is the solution. In addition it is complete: if there exists a solution then there is an

execution trace that will always return this solution plan. The solution plan is returned in time linear to the number of steps in the execution trace; for all practical purposes this is a reasonable amount of time – limited to one to two seconds in our experimental evaluation.

On the other hand this approach of problem solving lacks heuristics and it is not considered as the optimal one in terms of time, since the number of applicable actions in each state can be large or it can generate a large part of the search space which makes it highly inefficient. However, the way our system is defined restricts the applicable actions (messages) in every state to be exactly *one* while the number of the possible actions and states is very small. So choosing this kind of implementation seemed reasonable since it serves our purposes and does not complicate our system.

### 5.3 Communication System

The agent communication system follows the same implementation as the previous version of ORS: it is based on the *Linda* process communication. Linda is a set of tuple space language extensions; the version used in our project is written in Sicstus Prolog. The basic idea followed is that one process runs as the server while one or more other processes run as the clients. The basic predicates used by the agents are the *out/1*, *in/1*, *in – noblock/1* and *in – block/1* in order to send a message or to read a received message.

The communication protocol that is already specified in the existing ORS defines two different performatives. The first one, is *query* which is used by the agents to ask information. The other one is *reply*, which is used by the agents to provide information. So according to these performatives the agents are only enabled to ask and/or to reply to questions. The general format of the exchanged messages is:

`query(SendingAgent,ReceivingAgent,QueryType,Query)`

`reply(SendingAgent,ReceivingAgent,QueryType,Answer) .`

`SendingAgent` is the name of the agent that sends the message, `ReceivingAgent` is the name of the agent that this message is addressed to. `QueryType` refers to the issue this query addresses, and is categorised into *request*, *question*, *repair* and *repair2*. *Request* is used when an agent asks from another to perform a task, *question* is used for asking information i.e. about the requested task. Finally *repair* and *repair2* are used by an existing negotiation protocol implemented by [Bomersbach, 2011].

Query refers to the information exchanged regarding the `QueryType` while Answer

is the response of a query which can be some information needed, or one of *ok*, and *problem* which are predefined responses that map to a successful completion of a requested task, or to a problem regarding the requested task.

This protocol focuses on exchanging queries and information about the requested tasks, while it also fulfills the purposes of an existing negotiation protocol which is hard-coded in the agents.

The communication process that our system follows, is the same communication protocol as in the previous versions; apart from the messages used by the previous negotiation system. In addition we added two new query types: *negotiation* and *expert*. Negotiation is used during the negotiation process while expert is used for the exchanged messages that address issues regarding the expert agent's advice. In more detail the messages are of the form:

```
query(SendingAgent,ReceivingAgent,negotiation, [NegotiationInfo])
reply(SendingAgent,ReceivingAgent,negotiation, [NegotiationInfo])
```

These messages have the same form as previously defined, apart from the query type and the query itself. All the messages that are of the type negotiation include a list that contains negotiation related information. This information is entirely based on the negotiation ontology, so in order for the agents to interpret and reason these messages may have to consult the negotiation ontology. Such information can be the negotiation object, the negotiation messages and the effects of the negotiation messages.

The agents use the *NegotiationInfo* for consulting the negotiation ontology, which constitutes the precondition set for their response. If an agent cannot find the appropriate response in the negotiation ontology, it asks for more information, which will be the new precondition set for its response. For proceeding to this operation, the agents must send messages which concern the same negotiation object. For this reason when an agent consults the negotiation ontology, it searches for the message it has received, the effects of this message and it replies if and only if the negotiation object is the same for both agents.

In addition to the negotiation, we also introduce the expert query type. The messages concerning the expert query type are of the form:

- query(SendingAgent,ReceivingAgent,expert,[ExpertInfo])
- reply(SendingAgent,ReceivingAgent,expert,[ExpertInfo])

These messages have the same form as previously defined, and they are used by the agents when expert advice issues arise. The list *ExpertInfo* contains information regarding a request about asking the expert's advice. Such information can be a request by itself, or information relevant to an expert's advice, e.g. the suggested representation and the agent who should perform the repair, a request about following this advice or a reply constituted by a *yes* or a *no*, regarding the previous requests.

The reason we introduced this type of query is because we aimed to provide the agents with the freedom to discuss the expert's advice beyond the negotiation purposes. Such implementation enables the agents to come to an agreement and to ask for the expert's advice before the negotiation starts. This could be beneficial for both the negotiation process but also for the agents individually since they can use this advice for their own interests (see future work).

In more detail, the exchanged messages during the negotiation process are of the form:

- `query(Agent1,Agent2,expert, [ExpertInfo]),`
- `reply(Agent1,Agent2,expert, [ExpertInfo]) .`

Our main goal is not to restrict the agents on following a specific communication protocol designed for the negotiation purposes, which is achieved apart from these extra query types. The existence of this extra message is justified by the difficulties we faced on trying to achieve both of our goals at once: on the one hand not to hard code the agents and on the other hand to enable them to negotiate. *Negotiation* is introduced to the agent communication protocol as an effort to distinguish the messages which are sent and received until now from the negotiation messages. In addition, since we do not consider the expert's advice as a feature of our negotiation protocol, we believe that the expert query type does not have any side effects into our goal of not explicitly coding the negotiation protocol to the agents. However, we still embrace the idea of not hard coding the agents and we suggest that adding a single query type does not contradict it.

### 5.3.1 Processing the negotiation ontology

After the participants obtain the negotiation ontology, they consult it in order to interpret and reason about the exchanged messages. Before the negotiation begins, each agent is assigned a negotiation role: initiator or participant. This role constitutes one

of the preconditions of the way the messages are exchanged. Having as initial state the negotiation phase start, the agents proceed according to the plan that is included in the negotiation ontology. The initiator always starts the negotiation by searching the ontology and finding the message that initialises the negotiation.

Once the first negotiation message is sent, every time an agent receives a negotiation message of the form we defined in the previous section, it reasons about the negotiation information it has received using the negotiation ontology. The negotiation message, the effects of this message and the agent's assigned negotiation role form the preconditions of the agent's response. So it searches the ontology to find a rule that satisfies these preconditions. If no applicable rule is found, the agent requests more information regarding the last message it received.

Let us demonstrate a part of a negotiation between the PA and the *loanMoney* agent:

PA: The negotiation ontology has been consulted. I am assigned with the role of the initiator, therefore I am starting the negotiation. I am sending a call for proposal to the loanMoney agent, for the negotiation object switch arguments

loanMoneyAgent: The negotiation ontology has been consulted. I am assigned with the role of the participant. I received a call for proposal for the negotiation object switch arguments. I replied by sending a receive call.

PA agent: I received a receive call by the loanMoney. I am asking more information about loanMoney decision

loanMoneyAgent: I was asked to provide more information about my decision. I am sending a propose for the negotiation object switch arguments.

PA: I received a propose for the negotiation object switch arguments. replied yes . I am accepting this proposal.

Every time a message is exchanged, the receiving agent checks the negotiation ontology about the existence and the meaning of this message and replies according to the role it is assigned. As we can see in our example, when *loanMoneyAgent* sent a receive call to the PA, the PA asked for more information. This happened because the meaning of this message specified in the negotiation ontology does not provide to the

receiving agent the essential information about the sending agent's decision. Thus, the receiving agent cannot perform any further action until this decision is clarified.

During the negotiation process, both agents are able to abort on demand. This procedure repeats until the negotiation phase end is reached, while in every negotiation round the ontology the agents consult is a different one.

The negotiation reasoning process the agents follow can be thought as a very simple planning process. This is justified since the negotiation ontology contains a set of rules, and the agents try to find the next applicable message, based on these rules and the exchanged messages.

## 5.4 Forming the utility

The approach of forming the utility that we follow in our system is protection based. This means that once the agents are aware of the suggested repair they check their ontological part this repair deals with for any kind of protection. Based on the protection level, if any, they form their utilities.

From the previous negotiation system, the agents were provided with an algorithm for checking their utilities for any kind of protection. After this check they performed the repair. Having this algorithm as a basis we modified it in a way that we:

- disabled the repair performing part, and
- we added the utility function.

In this way, the agents form their utilities and they do not proceed to any kind of repair until an agreement is reached.

As an example, in the previous system the PA would check its ontology for protection and in case there was no protection it would implement the repair. In contrast to this, in our system the PA checks its ontology for protection, and in case that it has no protection it forms its utility accordingly and then proceeds to the negotiation part. We consider this alteration of the algorithm advantageous, since it enables the agents to negotiate about the repair even if their ontological parts are not protected. One benefit from this implementation is that we consider cases in which the cost of a repair is not totally expressed by the ontology protection but from other factors as well. In such cases we assume that the agents prefer to negotiate first than instantly perform the repair.

We chose all the agents to form their utilities based on their protection levels. However this approach does force the agents to follow only this kind of utility forming. Since the agents are autonomous, they can choose their own way of forming their utilities which can be based on different factors.

## 5.5 Summary

In this chapter we have described the main principles followed for the implementation of our system. We believe that our implementation approach provides the flexibility for successfully integrating our system with others that follow different negotiation approaches and protocols. The way the negotiation system is implemented can be thought of as a plug in to the agents, and can easily be replaced by another one or to be extended by adding more negotiation protocols or different planning approaches (see future work).

# Chapter 6

## Evaluation

### 6.1 Aims of the Evaluation

This project aims to implementing a negotiation protocol that facilitates the interaction of the agents in a more sophisticated way than the previous attempts. Our basic goal is to increase the agents autonomy regarding the repair allocation, by introducing a sophisticated way of negotiation. This results in the agents avoiding any unnecessary repairs. The existing versions of ORS are three: the first version of ORS and two more extensions of that version. The first version of ORS does not concern with ontology protection nor with any kind of negotiation; the PA always performs the repair. Since the outcome of this system is known beforehand (the PA will always perform the repair) we decided not to include this system in the evaluation tests we conducted. The second version of ORS is concerned with ontology protection and which we refer to as the original version of ORS. At this point we should clarify that this version of ORS operates in the same way as the first version; the only difference is that this version deals with ontology protection. The PA is the only agent that has ORS as a plug-in and is the only agent concerned with the repair. Since this version deals with ontology protection, the PA will perform the repair only if it does not have any kind of protection for this repair. Based on this fact it seemed reasonable not to compare the outputs of our system with those from the original ORS, but instead to observe the plan execution procedure in both systems. This is going to be the evaluation of our hypothesis:

It is possible to introduce a more sophisticated interaction between the agents than in the original ORS, by enabling the agents to negotiate over repair allocation. Throughout all this we guarantee that plan execution proceeds at least as well as in the original ORS.

The third version of ORS [Bomersbach, 2011] is an extension of the original ORS which deals with negotiation over repair allocation. So, based on our hypothesis, our project is characterised as successful in cases that the plan execution proceeds at least as well as in the original ORS. However, since the third version of ORS deals with repair allocation and negotiation we considered it to be essential to perform a comparison between the negotiation outputs of our system and the outputs of the previous negotiation system. The protocol that the previous negotiation system follows is that a negotiation will take place if and only if the PA has protection for the repair. The negotiation begins with the PA requesting from the SPA to perform the repair. The SPA will agree to perform it only if it has no protection at all, else it will refuse. If the PA receives a refusal and it has low protection it will perform the repair; otherwise it requests again from the SPA to perform the repair. The SPA will only agree to perform the repair if the the protection it has is low.

The evaluation of our system is decomposed to:

- Evaluation of our system in terms of plan execution. We expect our system to perform at least as well as the original ORS (which does not include any kind of repair allocation). As a good performance of our system we mean that the plan execution does not fails more often than in original ORS.
- Negotiation process outcome, in which we compare the outcomes of our system with the previous ones using the same scenarios.
- Negotiation Protocol Evaluation: We evaluate the protocol in terms of: the communication effort that is needed, deadlocks occurrence, negotiation process termination, *pareto optimality* of the agreement and the implementation of the protocol.

In the following sections we describe the evaluation process we followed as well as the results we obtained.

## 6.2 Negotiation Process evaluation

For the experiments we conducted, we ran in all three systems the same scenario and compared the results we obtained. We created one new scenario, and we also used one that was used in the evaluation of the previous versions of ORS. In addition we evaluated the performance of our system using one of the existing scenarios (from

previous projects). In this way we ensured that our system is capable of executing scenarios that were not implemented to serve the exact purposes of our project.

Ontology protection for both the PA and the SPA is considered in both our systems and in the previous negotiation system. In the original ORS, since the PA is the only agent that performs the repairs, the system concerns only with the PA's ontology protection.

### 6.2.1 The Economy scenario

The first set of tests are conducted based on the scenario that was implemented for this project: the economy scenario. The economy scenario is a satiric simulation of how a country can end up with loans. In this scenario, the PA plays the role of the Greek prime minister. The PA interacts with three service providing agents: *acceptEuropeanUnionAgent*, *giveEuroAgent* and *loanMoneyAgent* in order to ask them to perform the necessary tasks that will lead to the success of its plan. Each of these SPAs can perform the task the its name declares. *AcceptEuropeanUnionAgent* can perform the action *acceptEuropeanUnion(country, continent)*, *giveEuroAgent* can perform the action *giveEuro(continent, country)* and *loanMoneyAgent* can perform the action *loanMoney(country)*.

For each of the individual agents we created two ontologies: the main ontology which contains ground information about the domain, and the meta ontology which constitutes an ontology about the basic ontology. In the meta ontology we included the protection specifications we used for every agent.

The PA takes as input the goal *withMoney(greece)* and produces a sequence of actions, the plan that will enable it to succeed this goal. The plan that is produced for this specific goal is [*acceptEuropeanUnion(europe, greece)*, *giveEuro(europe, greece)*, *loanMoney(europe, greece)*]. For each of these actions, the PA will request the corresponding SPA to perform it.

The execution of the plan starts with the PA asking the *acceptEuropeanUnion* agent to perform the *acceptEuropeanUnion(europe, greece)* task. After receiving the request, *acceptEuropeanUnionAgent* checks its preconditions for this task and asks the PA about these preconditions. *AcceptEuropeanUnionAgent* will perform the requested task only if all the preconditions defined in its ontology match with those in the PA's ontology. So, in order to check if these preconditions match, it questions the PA about them. The dialogue of the two agents is:

PA: I requested *acceptEuropeanUnionAgent* to perform *acceptEuropeanUnion(europe, greece)* for me.

*acceptEuropeanUnionAgent*: I received a query about performing the task *acceptEuropeanUnion(europe, greece)*. I want to perform *acceptEuropeanUnion(europe, greece)* for the PA. I must check my preconditions... I have asked the PA about the precondition *located(europe, greece)*.

PA: I was asked about *located(europe, greece)*. I replied no.

*acceptEuropeanUnionAgent*: The PA replied no. Sorry, I cannot perform this task.

After this failure, ORS takes as inputs the exchanged messages and diagnoses the cause of the failure and suggests a repair in order for the agents to continue their interaction. In this case, it finds that the arguments are in different positions and suggests the repair switch arguments. The consequence of the PA repairing its ontology would be  $located(greece, europe) \mapsto located(europe, greece)$ , while for *acceptEuropeanUnionAgent* would be  $located(europe, greece) \mapsto located(greece, europe)$ . After the repair is performed and the agents' representations are aligned, the agents can continue their interactions so the PA to proceed to its plan execution.

One kind of protection that is checked for this repair is the whole predicate protection (for the predicate *located*), which we are going to use in our test cases.

This is the framework for test case A. We run this scenario on both our negotiation system and the previous one, while the results from the original ORS are empirically obtained.

### 6.2.1.1 Economy Scenario test case A.1

For this set of tests, none of the agents' predicates are protected, so consequently both agents have utility equal to *two* for this repair. The representation that the expert agent suggests as the correct one is the *located(greece, europe)*. The agents in this test do not agree to ask for the expert agent's advice.

After running this scenario to all three versions of ORS the results we obtained were:

- original version of ORS: the PA performed the repair while no negotiation takes place.

- Previous Negotiation version of ORS: the PA performed the repair while no negotiation took place.
- Our negotiation system: *acceptEuropeanUnionAgent* performed the repair as an outcome of a complete negotiation process, the agents did not agree on consulting the expert agent.

As we can see in the results, in both older versions of ORS it is always the PA that performs the repair. This is not considered a problem for this scenario, since none of the agents have any kind of protection. In addition, as the results show, in the previous version of ORS the agents do not proceed to any kind of negotiation. This is because the system is implemented in a way that in cases the PA does not have any kind of protection it does not proceed to a negotiation for the suggested repair; it instantly performs it. However, lack of protection does not imply that this repair does not have any side effects for the agent that performs it. In our system, even in when that the PA does not have any kind of protection, it proceeds to a negotiation about this repair. In this case the PA preferred not to perform the repair, since its utility was not higher than the other agent's. We do not suggest that the outcome of *acceptEuropeanUnionAgent* performing the repair is better than the outcomes from the other systems, but we do believe that we offer more flexibility and a more sophisticated way for the agents to decide about repair allocation.

In the second test we conducted the agents asked for the expert agent's advice and they also agreed to follow it. The results we obtained were:

- original version of ORS: the PA performed the repair while no negotiation took place.
- Previous Negotiation version of ORS: the PA performed the repair while no negotiation took place.
- Our negotiation system: *acceptEuropeanUnionAgent* performed the repair as an outcome of a complete negotiation process. The agents obtained the expert agent's advice, which suggested *acceptEuropeanUnionAgent* for performing the repair and agreed to follow it. The PA incremented its utility by *one*, and the negotiation outcome was that *acceptEuropeanUnionAgent* perform the repair.

In this test the results about the agent allocated with the repair, are the same as in the previous tests. In our system, the agents consult the expert agent which suggests

*acceptEuropeanUnionAgent* making the repair, they follow this advice, and the outcome is that *acceptEuropeanUnionAgent* performs the repair. The difference between these results and the previous ones is the ability of the agents to consult and to follow the expert agent's advice.

In both tests of our system, plan execution was successfully completed and the goal was reached as in the original ORS. The difference in our system was that it enabled repair allocation, which was based on a more advanced way of deciding. In addition, the existence and the ability of the agents to consult and follow the expert's advice, as the third test shown, proves to be beneficial for the agents choosing a more efficient repair allocation. Based on these tests we conducted we concluded that in both previous versions of ORS it is the PA that always performs the repair, while in the previous version of the negotiation system the negotiation never took place. We cannot condemn the outcomes of the previous versions, but taking into consideration that we intentionally misrepresented the *acceptEuropeanUnionAgent* agent's precondition, he have good reasons to reprove these outcomes. In the last case the the agents do not only decide based on their utilities but also based on the expert agent's advice which is aware of the correct representation. In this case we can consider our system more beneficial, since besides from enabling repair sharing, it provides a more advanced way of deciding repair allocations. Finally it provides all the means to the agents to come to the best possible agreement.

#### 6.2.1.2 Economy Scenario Test case A.2

For the following sets of tests we assigned both agents a low predicate protection (consequently their utilities are *one*) while the representation that the expert agent suggests as the correct one is located(*greece, europe*).

In the first test we ran, the agents did not agree to consult the expert agent's opinion. The results we obtained are:

- original version of ORS: the PA did not perform the repair and the plan failed. No negotiation took place.
- Previous negotiation system: the PA performed the repair while a simple kind of negotiation took place: The PA asked twice *acceptEuropeanUnionAgent* to perform the repair, which both times replied negatively.
- Our negotiation system: *acceptEuropeanUnionAgent* performed the repair as an

outcome of a complete negotiation process. The agents did not agree to consult the expert agent.

In this case, in the original ORS, the PA does not perform the repair because of the low protection, and the plan fails. In the previous negotiation system, the PA sends twice the same request to *acceptEuropeanUnionAgent*, about performing the repair. *AcceptEuropeanUnionAgent* refuses to perform it, because of the low protection, so the PA finally performs it. In our system, after the agents exchanged their will to consult the third agent (they did not come to an agreement), a complete negotiation round took place in which *acceptEuropeanUnionAgent* proposed to perform the repair and the PA agreed. In this case we consider our system more advanced than the previous negotiation system, since we provide the agents with the ability to come to a mutual agreement according to their own interests (utilities and expert advice).

In the second test we ran, the agents agreed on consulting the expert agent's opinion but did not agree on following it. The results we obtained are the same as from the previous test, so for this reason we are not going to analyse them further. Finally in the third test we conducted the agents agreed to consult the expert agent and also to follow its advice.

- original version of ORS: the PA did not perform the repair and the plan failed. No negotiation took place.
- Previous negotiation system: the PA performed the repair while a simple negotiation took place: The PA asked twice *acceptEuropeanUnionAgent* to perform the repair, which both times replied negatively.
- Our negotiation system: *acceptEuropeanUnionAgent* agent performed the repair as an outcome of a complete negotiation process. The agents agreed to consult the expert agent and followed this advice.

In this case having as preconditions the agents' utilities and the expert agent's advice, the previous negotiation system did not manage to proceed to a *fair* repair allocation. We use the term fair for referring to an allocation that respects both the agents' utilities and the expert agent's advice which is considered correct. Our system took into consideration all these aspects and concluded to *acceptEuropeanUnionAgent's* allocation. This repair has side effects for both agents so we consider this allocation

fair as the agent that is believed to have the wrong representation is chosen. In contrast, in the previous negotiation system the PA repairs its ontology based on a wrong representation.

In all three cases our system managed to complete the plan and to reach the goal. Compared to the original version of ORS, where the plan failed, we managed to succeed in two goals at once: the plan is successfully performed and the PA does not perform the repair which was the cause of the plan failure in the original ORS.

### 6.2.1.3 Economy scenario test case case A.3

In the following cases we will test all three systems, assigning both agents with high protection (consequently their utilities are *zero*). In addition the representation that the expert agent considers as the correct one is the *located(greece, europe)* which suggests that the PA's representation is the correct one. In the first two scenarios, in which the agents did not follow the expert agent's advice all systems failed to complete the plan execution. We cannot consider these cases as failures, since it is reasonable from the agents to prefer to keep their ontologies safe than performing the repair and continue the plan. We respect the agents preferences and we do not force them to perform the repair. In the third case the agents chose to follow the expert's advice, and the results we obtained are:

- original version of ORS: the PA did not perform the repair and the plan failed. No negotiation took place.
- Previous negotiation system: none of the agents performed the repair and the plan failed. A small of negotiation took place: The PA requested twice from *acceptEuropeanUnionAgent* to perform the repair which both times replied negatively.
- Our negotiation system: *acceptEuropeanUnionAgent* performed the repair as an outcome of a complete negotiation process. The agents agreed to consult the expert agent and followed this advice.

In this case our system is the only one that succeeded to complete the plan and to reach the initial goal. The agents chose to follow the expert agent's advice and finally came to an agreement about the *acceptEuropeanUnion* agent performing the repair. This set of tests proves that the existence of an expert agent can be very beneficial for

the system while the flexibility our system offers enables the agents to take advantage of this feature.

#### 6.2.1.4 Economy scenario test case A.4

For the following set of tests we assigned the PA with high protection and *acceptEuropeanUnionAgent* with no protection at all. Consequently the PA's utility is *zero* while *acceptEuropeanUnionAgent's* utility is *two*. This time we are going to use a different suggested representation by the expert, which is *located(europe, greece)*. The results we obtained are below.

In the first test scenario:

- original version of ORS: the PA did not perform the repair and the plan failed. No negotiation took place.
- Previous negotiation system: *acceptEuropeanUnionAgent* performs the repair while a simple negotiation took place: The PA asked once the *acceptEuropeanUnion* agent to perform the repair and the *acceptEuropeanUnion* agent accepted.
- Our negotiation system: *acceptEuropeanUnionAgent* performs the repair as an outcome of a complete negotiation process. The agents did not agree to consult the expert agent.

In this test, our system managed to successfully complete the plan in comparison with the original ORS that failed. Both in our system and in the previous negotiation system *acceptEuropeanUnionAgent* is assigned with the repair and after the repair was performed the PA continued with its plan execution. We obtained the exact same results for the second test in which the agents consulted the expert agent but did not follow its advice. In the third test, the agents followed the expert agent's advice. The results are:

- original version of ORS: the PA did not perform the repair and the plan failed. No negotiation took place.
- Previous negotiation system: *acceptEuropeanUnionAgent* performs the repair while a simple negotiation took place: The PA asked once *acceptEuropeanUnionAgent* to perform the repair and *acceptEuropeanUnionAgent* accepted.

- Our negotiation system: *acceptEuropeanUnionAgent* performs the repair as an outcome of a complete negotiation process. The agents agreed to consult the expert agent and followed this advice.

Although this experiment gave us the same results as the two previous ones, the reason we analyse it is because of the agents' behaviour towards the expert agent's advice. Although the agents followed the expert agent's advice, which suggested the PA performing the repair, as we can see the agent that finally performed it was *acceptEuropeanUnionAgent*. This happened because the agents do not entirely base their agreement on the expert agent's advice. This advice changes the outcome of an agreement only in cases that the agents' utilities are equal. The agents firstly aim to protect their ontologies and then to follow the expert advice. Thus, the advice does not strongly influence the outcome.

This set of tests, proved that our system is capable of succeeding in plans that the original ORS would fail.

## 6.2.2 Water Management scenario

The water management scenario was implemented for evaluating the previous negotiation system. It deals with water treatment in order to maintain the best quality of water. In this scenario, the PA interacts with the *plantOne*, *plantTwo*, *plantThree*, *plantFour* and *plantFive* agents (depending on the test scenario), in order to achieve the goal *goodPerformance(plantZero)* (first test case) or the goal *withChlorine(plantZero)* (second test case). In the following sections we present and analyse the results we obtain from the experiments based on this scenario.

### 6.2.2.1 Water management scenario Test case A.1

For the following experiment, the PA produces a plan in order to achieve the goal *goodPerformance(plantZero)*. The plan that is produced is the following sequence of actions: [*sendContaminatedWater(plantOne, plantZero,x,perfect)*, *treatContaminatedWater(plantZero, x, perfect)*]. The PA begins this plan execution by requesting the service providing agents, *plantOne* and *plantThree*, to perform these two actions for it. The first request the PA sends is a request about the action *sendContaminatedWater(plantOne, plantZero,x,perfect)* to agent *plantOne*. *PlantOne* in its turn, in order to perform this requested task, must check whether the preconditions of this task in its ontology match the preconditions in the PA's ontology. This is achieved by asking the

PA about these preconditions. So *plantOne* starts asking questions and eventually the communication fails. This happens because of the different representations these two agents have of the precondition *treats*. The PA's representation is *treats(plantZero, x, perfect)*, while *plantOne*'s representation is *treats(plantZero, x)*.

After the communication failure, ORS takes as inputs the messages the agents exchanged and diagnoses the different arity of these two predicates to be the cause of the failure. The suggested repair for this case is propositional abstraction for the PA or propositional refinement for the SPA. For this test case we assigned *plantOne* with high whole argument protection at position three, which is a kind of protection that is checked for the specific repair, while we assigned the PA with no protection at all. The expert agent's advice suggests that the correct representation is *treats(plantZero, x, perfect)*.

We ran this scenario twice: the first time the agents did not follow the expert agent's advice, while the second one they did. The results we obtained from these tests cases are

- original version of ORS: the PA performed the repair. No negotiation took place.
- Previous negotiation system: the PA performed the repair. No negotiation took place.
- Our negotiation system: the PA performed the repair as an outcome of a complete negotiation round. This is the same outcome for both following and not following the expert agent's advice.

In both cases the negotiation outcome was the PA performing the repair. This test results give us an example of how much impact the expert's advice has on the system. Since the *plantOne* agent had *zero* utility for this negotiation round, the expert's advice did not change the outcome of the negotiation.

In all three systems the plan was executed successfully, while in all three cases the PA was the agent that performed the repair. This can be justified by lack of protection on the PA's ontological object. In the original ORS the PA always performs the repair in case of no protection, and the same idea is followed by the previous negotiation system: in case of no protection the PA always performs the repair without starting any kind of negotiation with the SPA. In our system, the PA will always negotiate the repair regardless of its ontological protection. Although the negotiation outcome of our system is the same repair outcome of the other systems, we consider in this case

our approach more advanced in the sense of providing the PA with the freedom to negotiate over the repair. By negotiating the repair, the PA increases its possibilities of not performing it which otherwise are zero.

In this test case our system managed to execute the plan as successfully as the original ORS.

### 6.2.2.2 Water management scenario Test case A.

For this set of tests, the goal that the PA aims to achieve is *withChlorine(plantZero)*. The plan that is produced for this goal is *[sendWater(plantZero, x), sendChlorine(plantZero, x)]* and the agents that the PA interacts with are *plantTwo* and *plantFive*. Initially the PA asks *plantTwo* to perform the actions *sendWater(plantZero, x)* for it. When *plantTwo* receives the request it checks whether its preconditions match those of the PA. If they do, it performs the requested action; otherwise their communication fails. In this case the PA and the *plantTwo* agent have different representations of the precondition so their communication fails. The PA's representation is *at(plantZero, x)* while the corresponding representation of the *plantTwo* agent is *at(x, plantZero)*.

After the communication failure, ORS takes as inputs the messages the agents have exchanged and diagnoses that the arguments of the predicate *at* are in different positions, and suggests the repair switch arguments for the PA or for the SPA. For this test case we have assigned the PA with whole predicate high protection, which is the kind of protection that is used for the switch arguments repair, and the *plantTwo* agent with no kind of protection. In addition, the representation the expert agent advises as the correct one is set to be *at(plantZero, x)* which implies that the *plantTwo* agent should perform the repair.

We ran this test scenario for both the case in which the agents do not follow the expert agent's advice, and for the case they do. The results we obtained are:

- original version of ORS: plan failed
- Previous negotiation system: the *plantTwo* agent performed the repair. A simple negotiation took place.
- Our negotiation system: the *plantTwo* agent performed the repair as an outcome of a complete negotiation round. This is the same outcome for both following and not following the expert agent's advice.

As we can see, in this case the original ORS fails to execute the plan because of the PA's high ontological protection. On the other hand both our system and the previous negotiation system achieve to complete the plan, which indicates that repair allocation outperforms the continuous repair performance of the same agent. The process the previous negotiation system followed was a simple request from the PA to the plantTwo agent for performing the repair. Instead in our system, a whole negotiation process took place in addition to the decision process of whether the agents should follow or not expert's advice. So, the difference between the two negotiation systems is the interaction richness between the agents during the negotiation, in addition the possibility of obtaining the expert agent's advice.

In this test scenario, our system managed to successfully complete the plan execution, while the original ORS failed.

### 6.2.3 Timings

In addition to the evaluation we described in the previous sections, we also compared the execution times of these three systems for each scenario. We chose to run two test cases for each scenario, one in which the PA had no protection for the suggested repair and one in which the PA had protection. The reason we chose to conduct two different test cases for each scenario was that we wanted to obtain results from both cases in which negotiation takes place as well as from cases that there is lack of negotiation. Since in the original ORS the plan execution succeeds only if the PA has no protection for the suggested repair it was essential one of the test cases to fulfill this requirement. However, in the previous negotiation system the protocol that is followed suggests that in case that the PA has no kind of protection there will not be any negotiation about the repair allocation. Since we wanted to compare the execution times of both negotiation systems (our and the previous one) it was essential to run a scenario in which the agents negotiate over the repair allocation. The results we obtained are shown in Figure 6.1 in the form of chart bars.

For the economy scenario we ran two test cases: in the first one, for the repair *switch arguments* the PA did not have any kind of protection while the SPA had high. In the second test case scenario both the PA and the SPA had low protection. The first case (test case 1) was chosen is because it offers us a scenario in which the original ORS succeeds in executing the plan. Both the original ORS and the previous negotiation system have the same execution times since in both systems the exact same operations

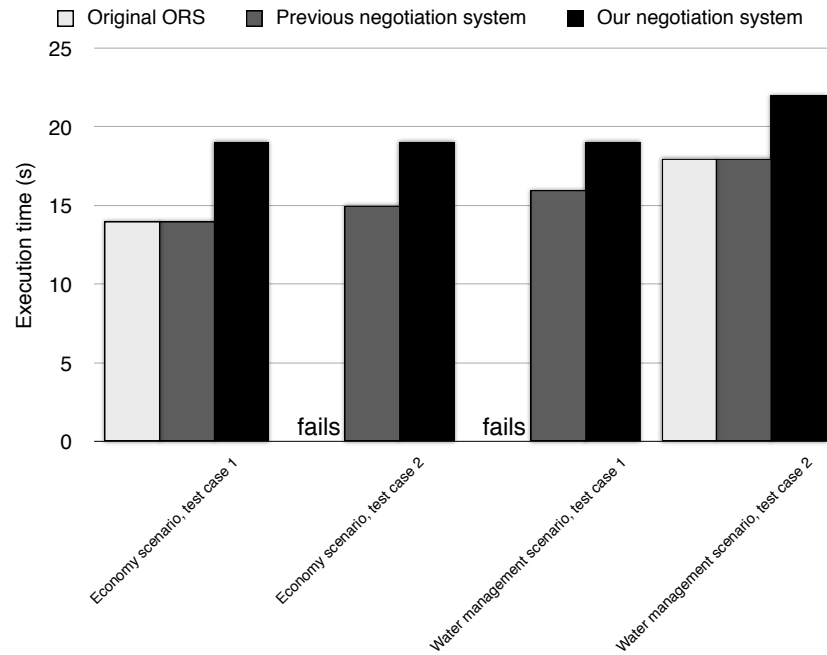


Figure 6.1: Execution times of the three ORS versions

take place (due to the lack of negotiation). So, for this reason we chose to run and the second test scenario (test case 2) (in the previous negotiation system the agents negotiate over the repair allocation), in which the original ORS failed to complete the plan (since the PA had protection).

For the water management scenario we also conducted two different tests. In the first test case the PA had high protection for the repair *switch arguments* while the SPA did not have any kind of protection. In the second test case the repair is *propositional abstraction*, the PA had no kind of protection for this repair while the SPA had low protection. In the first case scenario (test case 1) the original ORS failed to complete the plan (because of the PA's protection) so we only compared the time execution of the previous negotiation system and the time execution of our negotiation system. For the second test case (test case 2), both the original ORS and the previous negotiation system have the same execution time because in both systems no negotiation takes place.

The overall results we obtained have shown that our negotiation system does not pose a large overhead on the original ORS. We believe that the performance overhead

is a small price to pay compared to the benefits our negotiation system offers.

## 6.2.4 Summary

In the previous sections we presented a series of tests we conducted in order to evaluate our system's performance. We have conducted a total number of fifteen test cases for each scenario, in which we have covered all the possible combinations of protection levels and expert agent's advice. The repair outcomes of the 15 conducted tests are shown in Tables 6.1, 6.2, and 6.3. ORS stands for the system that do not follow any kind of negotiation. Symbol "-" indicates that no repair is performed and that the plan failed. When a repair is performed the plan always succeeds.

test scenario case	Our system	Previous negotiation system	ORS
PA low protection and SPA low protection	SPA	PA	-
PA low protection and SPA high protection	PA	PA	-
PA low protection and SPA no protection	SPA	SPA	-
PA no protection and SPA low protection	PA	PA	PA
PA no protection and SPA high protection	PA	PA	PA
PA no protection and SPA no protection	SPA	PA	PA
PA high protection and SPA low protection	SPA	SPA	-
PA high protection and SPA high protection	-	-	-
PA high protection and SPA no protection	SPA	SPA	-

Table 6.1: The agents did not follow any expert agent's advice.

test scenario case	Our system	Previous negotiation system	ORS
PA low protection and SPA low protection	PA	PA	-
PA low protection and SPA high protection	PA	PA	-
PA low protection and SPA no protection	SPA	SPA	-
PA no protection and SPA low protection	PA	PA	PA
PA no protection and SPA high protection	PA	PA	PA
PA no protection and SPA no protection	PA	PA	PA
PA high protection and SPA low protection	SPA	SPA	-
PA high protection and SPA high protection	PA	-	-
PA high protection and SPA no protection	SPA	SPA	-

Table 6.2: The agents followed the expert agent's advice which suggested the PA to perform the repair.

The evaluation we followed, justified our hypothesis that by introducing to the existing ORS a negotiation protocol we enable agent negotiation over repair allocation in

test scenario case	Our system	Previous negotiation system	ORS
PA low protection and SPA low protection	SPA	PA	-
PA low protection and SPA high protection	SPA	PA	-
PA low protection and SPA no protection	SPA	SPA	-
PA no protection and SPA low protection	SPA	PA	PA
PA no protection and SPA high protection	PA	PA	PA
PA no protection and SPA no protection	SPA	PA	PA
PA high protection and SPA low protection	SPA	SPA	-
PA high protection and SPA high protection	SPA	-	-
PA high protection and SPA no protection	SPA	SPA	-

Table 6.3: The agents follow the expert agent's advice which suggests the SPA to perform the repair.

a sophisticated and meaningful way while at the same time our system performs at least as good as the initial one. We have also illustrated cases where the original ORS fails to execute the plan due to the PA's protection while in our negotiation system the plan execution succeeds. We do not suggest that failing to execute the plan is a negative outcome, because there are cases in which protecting the ontology is more important than executing the plan. However, our negotiation system provides the ability to achieve both goals: succeed to execute the plan and avoid any ontology alterations in case of protection. As shown in the cumulative results of Tables 6.1, 6.2, 6.3, 6.2, and 6.3 our system achieves to successfully execute a plan in five or six extra times compared to the original ORS (the number of extra successful executions depends on the scenario).

In addition we have shown that our system enables the agents to come to a mutual agreement about the repair allocation. At the same time we provide the planning agent with the flexibility of avoiding to perform repairs that in the previous negotiation system would be have to. In more detail, in the previous negotiation system the PA will always perform the repair unless its protection is high. In our system both agents have equal possibility of performing the repair since the allocation depends entirely on their agreement.

### 6.3 Negotiation Protocol Evaluation

For the needs of this project we have designed a standardised infrastructure though which autonomous entities can communicate and reach an agreement in terms of a contract. The domain that the negotiation deals with by its nature restricts the ne-

negotiation protocol to be simple and straightforward. At each negotiation round the participants negotiate over the allocation of a single repair while the number of the negotiators is restricted to be equal to the number of the agents that were initially interacting. However the protocol is extensible, so more agents could be added (see future work Chapter 8).

The protocol that we have designed is a *one-to-one* task oriented protocol. It deals with task allocation with respect to the agents' preferences. We have expressed these preferences using the utility function which measures the risk an agent would take by performing the suggested repair by ORS. This risk is based on ontology protection; however our negotiation protocol does not restrict the way each individual agent chooses to express its preferences and is flexible enough to deal with any kind of different representations.

When we designed the negotiation protocol, apart from the ORS requirements, there were three additional requirements which we considered as essential for our protocol:

- **Communication effort is minimised.**

The negotiation protocol specifies nine permitted messages which can be exchanged by the agents during a negotiation round. These nine messages have been carefully designed to cover every possible case that can occur during a negotiation round. In addition we have limited the number of messages per transitions to be exactly one. This by its nature minimises the communication effort needed during the negotiation round. Using proof by contradiction we can easily prove this feature. As an example, consider that the current negotiation phase of our system is the negotiation phase *proposal-sent* and that we desire to transit our system to the negotiation phase *proposal-accepted*. Now, let us consider that instead of one message per transition, the system needs two. Recalling Figure 3.8, the exchange of two messages in a row will transit the system to the negotiation phase *end*, instead of the desired one. The same example can be illustrated by choosing any of the defined negotiation phases as the initial one, demonstrating that in order to transit to the next negotiation phase only one message needs to be exchanged. In addition, the messages are defined in such a way that the each participating agent is assigned with a specific set of messages that is permitted to send. In this way we make sure that the sequence of messages that occur during a negotiation round follow a meaningful order.

On the other hand, the definition of the smallest possible message set, restricts the richness of the dialogue and the information that can be exchanged during a negotiation process. However, the negotiation always comes after a previous interaction between

these agents, which continues after the negotiation. Therefore it is justifiable not to consider the lack of expressivity during the negotiation as a drawback.

- **A negotiation round never reaches a deadlock.**

In order to ensure that no deadlocks occur during the negotiation process, we have to make sure that the final state is always reached. For proving this property of our protocol we chose to model the whole negotiation process as a finite state model, and to check this property using Computational Tree Logic (CTL). CTL is used for formal verification and is usually used for checking models for undesirable properties.

For the purposes of this evaluation we used the NuSMV tool [Cimatti et al., 2000]. NuSMV is a BDD-based model checker that can process files written in SMV language and allows for the construction of models with different modalities, reachability analysis CTL or LTL (Linear Temporal Logic) model checking, and generation of counterexamples. NuSMV has an interactive mode through an interactive shell where it enters a shell performing read-level-print loop and the user can activate the various computation steps as system commands with The model checker executes the standard CTL algorithm and reports *true* if the property is satisfied by the model, or *false* if the property is violated. In the case of property violation NuSMV provides the routines for counterexample generations and inspection. Counterexamples can be produced with different levels of verbosity, in the form of reusable data structures, and can subsequently be inspected and navigated.

The input language of NuSMV is a description language that describes more what the program should accomplish than how it would. It considers programs in formal logic and computations as deductions in that logic space. Using this language we specified all the phases and the messages that are defined in the protocol, as well as the transitions the messages cause (see Appendix). The phases are represented as states and the messages as actions. For each state we have defined the applicable actions and the transitions these actions cause. For instance the phase *receive call* is represented as:

```
state = receiveCall : {callRefused , proposalSent;}
state = receiveCall & action = propose : proposalSent;
state = receiveCall & action = refuseCall : callRefused;
```

When a phase is specified, the first requirement is to represent the set of all possible next states. In this example, after the *receive call* phase either the *refuse call* phase or the *proposal sent* phase can occur. After the set of the next states is specified, every

possible transition is described. The only actions that can occur at this state are the *propose* and *refuse call*, so these are the transitions that have to be specified.

After we have modeled our system, we formed the following CTL property:  $AG (EF (state = end))$ . This property declares that the state *end*, which is the terminal phase of the negotiation process, is always reached. Checking this property in the NuSMV tool the console output that we got was:

*specification AG (EF state = end) is true*

which proves that the way the negotiation process is modeled, any kind of deadlocks are avoided.

Furthermore we have also checked two more properties which proved to be true:

- When a call for proposal is received, the receiving agent can always refuse this call or to make a proposal:

$AG (EF (state = callReceived \rightarrow EF (action = refuseCall \mid action = propose)))$

.

- When a proposal is received, the receiving agent can always either refuse it or accept it:

$AG (F (state = proposalSent \rightarrow F(action=refuseProposal \mid action = acceptProposal)))$

At this point we should clarify that we consider the communication between the negotiators reliable, like in the original ORS. We assume that the negotiators are honest and that the negotiation terminates when the final agreement is reached. We regard cases that an agent does not respond to a negotiation messages as a rejection of participating in the negotiation process.

• **Every possible agreement that is reached, maximises the social welfare, while is also *pareto optimal*.**

Let us recall the definition of the social welfare from Chapter 3: we define as social welfare the sum of the utility each negotiator has for an agreement. More formally, the social welfare for an agreement  $\delta_j(T_k)$  of agent  $j$  performing the repair  $T_k$  in a system of  $A_g = \{1, 2\}$  agents, is given by:

$$social\ Welfare(\delta_j(T_k)) = \sum_{i=1}^2 utility_i(\delta_j(T_k))$$

Moreover we consider that an agent that is not assigned with the repair has zero cost for this repair, so its utility is to *two*. Based on these definitions as we can see in

Table 6.4, the set of all possible agreements map to the maximum social welfare for every negotiation round.

utility combinations	Agreement reached	<i>social Welfare</i> ( $\delta_i$ )	<i>social Welfare</i> ( $\delta_j$ )
$utility_i = 1, utility_j = 0$	$\delta_i$	3	2
$utility_i = 1, utility_j = 2$	$\delta_j$	3	4
$utility_i = 2, utility_j = 1$	$\delta_i$	4	3
$utility_i = 2, utility_j = 0$	$\delta_i$	4	2
$utility_i = 0, utility_j = 1$	$\delta_j$	2	3
$utility_i = 0, utility_j = 2$	$\delta_j$	2	4

Table 6.4: The possible utility combinations that can occur and the negotiation agreement that is reached.  $\delta_i$  stands for an agreement of agent  $i$  performing the repair, while  $\delta_j$  stands for an agreement of agent  $j$  performing the repair

In addition we consider the agreement that is reached in each negotiation round, as pareto optimal since there is no other possible agreement that could benefit one agent without making the other agent worse off. This property is justified by the fact that each negotiation agreement that is reached is the only possible one that maximises the social welfare. In the case that there existed another agreement that could raise one agent's utility without decreasing the utility of the other agent, the social welfare of the first agreement would not be the maximum. But, recalling Table 6.4, this is not possible.

The only two cases that the agreement that is reached does not bring the maximum social welfare is when both agents have the same utility. In this case all the agreements belonging to the set of the permitted ones bring equal social welfare to the system. However, in this case none of the possible agreements dominates the other, so the agreement that is reached is still considered as *pareto optimal*.

### 6.3.1 Negotiation protocol implementation evaluation

The implementation we followed for the negotiation protocol implementation is an ontology-based approach. The reason we followed this approach was because we wanted to avoid any kind of hard-coding of the agents with the protocol. By creating an ontology that contains the negotiation protocol we facilitate communication between agents that are already hard-coded with different protocols. In this way we create a flexible system which can be easily adapted, extended, or substituted by a different one.

Such an implementation could only be tested by providing this ontology to agents that another protocol has already been made explicit to them. Fortunately, the previous negotiation system, which was implemented for ORS, followed the hard coding approach so it provided us a testbed for our implementation. The integration was achieved by creating a reasoning process, especially for the negotiation ontology. Since the SPAs were not provided with a planner, we created one. Every time a negotiation message is exchanged, this forms the precondition set and the agents use to find the correct response. However, we do not suggest that this is the only way to integrate a system with ours. If the agents are already provided with planner or another reasoning process for the ontology, the integration would be easier.

In order to achieve the integration of our system and with the previous negotiation system, we made available our negotiation service to the agents in the latter system. So when the agents wanted to negotiate, they invoked the negotiation process by providing their utilities and the negotiation object. The negotiation service then distributed the negotiation ontology to them. After the agents obtained the ontology, they consulted it and continued their negotiation based on the rules that this ontology contained.

The results showed that we managed to bypass the previous negotiation protocol. The agents negotiate successfully based on our protocol.

### **6.3.2 Summary**

In this chapter we presented the evaluation methods we used in our experiments in order to confirm our hypothesis as well as our implementation. We have proven that our system not only introduces a sophisticated interaction method for task allocation, but also manages to perform as well as the original ORS, for tasks that ORS can already handle. We have illustrated cases where the original ORS failed to complete the plan execution because the PA was incapable of performing the repair. These cases were used to provide examples in which negotiation not only increases the agents' autonomy, but also benefits the performance of the whole system.

In addition, we have compared the negotiation outcomes of our system to those of the previous negotiation system and have concluded that by defining a protocol we set ground negotiation rules, which make the whole process more advanced and straightforward. The protocol we have defined for negotiation needs covers all possible agent preferences. The previous negotiation system lacks this characteristic: cases in which the PA has no ontological protection imply lack of negotiation. In addition, by

introducing the notions of utility, cost and social welfare we have defined grounded justifications about the agents decisions during the negotiation process; in the previous negotiation system no such notions exist, while the way the agents decide was not straightforward.

Finally we have proved that our protocol is well-defined since it is never possible to reach a deadlock, it minimises the agent communication effort and it respects the negotiators' policies: every agreement that is reached maximises the social welfare while it is also *pareto optimal*.

# Chapter 7

## Related work

The continuous growth of the need for agent coordination and communication in distributed systems has led many researchers into focusing on the problem of ontology alignment and meaning sharing. While a great deal of research has been carried out to create the means for ontology mapping and alignment, the problem of whether and how these means can be used in open environments has not been much explored.

In the bibliography, one can find different approaches of how two interacting agents can align their ontologies, from which we chose the ones that are closer to the problem we are dealing with. In this section we discuss the efforts that have been made in order to dynamically solve the problem of ontology mismatches during agent interaction.

The work that has been done by [Avery and Yearwood, 2002] deals with the interaction between two agents:

1. the Agent, which has a known ontology, and
2. the Strange Agent, which has an unknown ontology.

The Agent identifies a Strange Agent when a communication failure occurs because of the ontological mismatches of their ontologies. After the communication failure, the Agent starts asking questions about the Strange Agent's representation and tries to reproduce it. The negotiation process is followed by a sequence of such reproductions by the Agent and the continuous feedback the Strange Agent provides. As an example, each time the Agent provides a representation, the Strange Agent either replies with a *yes*, or with a *no*. The agreement is reached when the Agent has successfully represented all the parts of the Strange Agent's ontology that caused the communication failure. The approach followed in this research is similar to ours: after a communication failure, the agents, through dialogue, try to learn more about their ontological

mismatches. This will be used afterwards for repairing their representations. However, the work described here entails that the Agent is the one that always aligns its ontology, which only partially solves the problem we are dealing with; there is no guarantee that the Strange Agent's representation is consistent or up to date. In contrast, in our approach we enable both agents to alter their ontologies while they have the freedom to choose which one is going to align its ontology with the other's.

Another negotiation-based approach is followed by [Schoop et al., 2003]. This work deals with how two agents can reach consensus about an ontology which will be later used for interaction purposes. Consensus is reached through negotiation. In more detail, two agents are provided with similar ontologies, while only one of these agents wants ontology alterations. The negotiation begins with this agent proposing to the other one the changes that it desires to perform. The reply to this request can either be positive, which results in both agents performing these alterations, or it can be negative, which means that no alteration takes place. Every proposal is made once, and when the requests are over, the outcome of the negotiation is a consistently adapted ontology. This entails that all the alterations are performed by both agents, while a precondition for this process is that both agents must initially have the same ontologies. On the one hand this approach deals with reaching consensus on ontologies, but on the other hand it does not suit the needs of open environments where it is more likely that the interacting agents have different ontologies. In contrast, our approach neither requires the agents to have the same ontologies, nor do we align the entire ontologies of the agents. Instead we deal only with the ontological parts that disturbed the agents' interaction.

The discovery and resolution of ontological mismatches was investigated in the work of [Bailin and Truszkowski, 2001]. An ontology negotiation protocol (ONP) is proposed as a means for finding ontology conflicts between agents and resolving them through negotiation. Interpretation, clarification and explanation are the steps that the agents follow during the negotiation process. In more detail, this protocol defines 46 times 46 possible negotiation transitions through which the negotiators exchange messages, determine whether a message is properly understood and evaluate a query that has been sent against the query itself. The negotiation outcome is that one of the two interacting agents will alter its ontology in order to introduce a new concept or a new term to an existing concept. This is a very interesting approach to the problem we are dealing with; nevertheless this protocol is applicable only in cases in which the ontologies of the agents provide more than a domain classification: it is essential

that synonyms and relations between domains are defined. This in itself restricts the protocol to be applicable in only a small amount of cases. Our problem domain is not one of these cases: we cannot be aware of the agents' representation beforehand, or whether this precondition is satisfied, so we cannot restrict the agents to follow a specific type of representation.

Finally, we discuss the work that is most relevant to our project: the previous negotiation system that was implemented for ORS needs. The negotiation approach followed by [Bomersbach, 2011] is based on ontological protection and proceeds between a planning agent and a service providing agent. In more detail, when an ontological mismatch that causes communication failure occurs between two interacting agents, the agents through dialogue, explore more about this mismatch. Based on this mismatch, ORS diagnoses the cause of the failure and suggests an appropriate repair that will enable them to continue their communication. When a repair is suggested, the PA checks its ontological parts for ontology protection, and if there is not any, it performs the repair. In case of protection, the PA asks the SPA to perform the suggested repair. The protocol that is followed is that the PA will ask the SPA to perform the repair. In case the response is negative, then it will ask again. The second response depends on the protection level this agent has. If the protection is low then the agent will *likely* reply positively. The positive response is likely and not certain because the service providing agent will only desire to perform the repair in case the planning agent's protection is high. The conclusion is that the PA usually performs the necessary repairs unless its protection is high. This approach fails into defining of what constitutes a legal agreement and how it is reached. In addition, no negotiation rules are clearly specified. In contrast to this approach, in our system the probability of each agent performing the repair is the same for all agents. The protocol defines what a legal agreement is and the possible ways this agreement is reached, the rules that govern the negotiation process, and every other aspect that can affect a negotiation decision or even the whole process. In addition, by defining the meaning of repair utility and social welfare, we ensure that the outcome of every negotiation round is *pareto optimal*.

To conclude, we have briefly presented research that has been done on some problem domain as ours. The problem can be seen from different perspectives so the solutions vary. We believe that our approach focuses more on the uncertainty that characterises open systems and provides an efficient solution that respects the agents' autonomy.

# Chapter 8

## Conclusions and future work

In this project we have designed and implemented a negotiation protocol for the needs of ORS. The negotiation that takes place is a task oriented one-to-one negotiation between the PA and the SPA. Every time a communication failure occurs between these two agents, ORS diagnoses the cause of the failure and, when an ontological mismatch is diagnosed, it suggests a repair. This repair constitutes the object over which the agents will negotiate. Before the negotiation starts, each agent forms its utility for this repair, which reflects the cost of making this repair, and then enters the negotiation. The protocol we have introduced to the agents assigns the role of the *initiator* to the agent that starts the negotiation, while the second agent is assigned with the role of the *participant*. In the implementation of this project, the PA was the agent that always started the negotiation because it was the agent that was informed first about the suggested repair. However this does not imply that this is the only way the negotiation can proceed. The protocol does not restrict which agent is going to be the initiator; it is always possible for the SPA to be the initiator.

Within the protocol we have specified the negotiation rules, the vocabulary, the set of the permitted negotiators and the set of legal agreements. The rules and the set of the legal agreements have been carefully designed to ensure that the negotiation outcome is always *pareto optimal*: the agents' preferences, which are expressed by the utility function, are always the priority of this protocol. However, this protocol is flexible to modifications and extensions. One possible extension would be to modify the negotiation from one-to-one to one-to-many. In this way, situations where the same repair has been suggested for more than two agents can be considered. Another possible extension would be to keep the history of all occurring negotiations. By doing this, agents that negotiate for a second time can query this history about their decisions.

The implementation we followed is ontology based, which means we avoided hard-coding the agents with the protocol. The implementation has thus far been tested by integrating the previous negotiation system with ours. This integration was achieved by slightly modifying the communication mechanism of the agents. Another interesting extension would be to include different protocols in the negotiation ontology. In this way, the agents can choose which protocol to follow or even suggest their own. Based on this idea another possibility would be to dynamically map any existing protocols that are hard-coded in the agents, into the ones offered by the negotiation ontology. So, when an agent acts based on its own protocol, the negotiation ontology can provide a translation from this protocol to the one which is hard-coded into the other.

In addition to the negotiation protocol, we have introduced the notion of the expert agent. This kind of agent is an entity with expert knowledge on the domain of the negotiation object. The negotiators can take advantage of this expert by asking for advice though it is up to the agents whether they will follow this advice. Currently we have introduced only one expert agent, so one of the possible extensions of this system could be the introduction of more experts, possibly with conflicting advice. These agents can exist in the system or can be suggested by the negotiators. This can prove an interesting research direction focusing on the way the agents choose which expert agent's advice to consider.

To conclude, in this project we have proved that repair negotiation is feasible in the scope of ORS without causing any side effects to plan execution. In addition the evaluation results indicate, that in some cases, when the original ORS fails to succeed in plan execution, our system succeeds. In addition we have shown the beneficial impact the expert agent has for the performance of the system. We believe that we have set the basis for an elaborate and sophisticated negotiation mechanism that can be further extended.

# Appendix A

## Economy Scenario

Below are the agent ontologies for the economy scenario. The goal for this scenario is *withMoney(greece)*.

### PA onto.in

```
;;; Agent
(Define-Class Agent (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Country
(Define-Class Country (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Money
(Define-Class Money (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Continent
(Define-Class Continent (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Sit-Var
(Define-Class Sit-Var (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Organisation
(Define-Class Organisation (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Euro
(Define-Individual Euro (Currency) "Not supplied yet.")
;;; Greece (Define-Frame Greece :Own-Slots ((Documentation "Not supplied yet.")
(Instance-Of Country)) :Axioms ((Located Greece Europe [(Start)]))
;;; Europe
(Define-Individual Europe (Continent) "Not supplied yet.")
;;; Europeanunion
```

```

(Define-Frame Europeanunion :Own-Slots ((Documentation "Not supplied yet.")
(Instance-Of Organisation)) :Axioms ((Have-Currency Europe Money [(Start)])))
;;; Pseudo-Var
(Define-Individual Pseudo-Var (Confirmation-Number) "Not supplied yet.")
;;; Sit-Var
(Define-Class Sit-Var (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Start (Define-Individual Start (Sit-Var) "Not supplied yet.")
;;; Indepted-Country
(Define-Relation Indepted-Country (?Country ?Situation) "Not supplied yet." :Def
(And (Country ?Country)(Sit-Var ?Situation)))
;;;Belongs
(Define-Relation Belongs (?Country ?Continent ?Situation) "Not supplied yet."
:Def (And (Country ?Country)(Continent ?Continent)(Sit-Var ?Situation)))
;;; Share (Define-Relation Share (?Country ?Money ?Continent ?Situation) "Not
supplied yet." :Def (And (Country ?Country) (Money ?Money)(Continent ?Continent)
(Sit-Var ?Situation)))
;;;Located
(Define-Relation Located (?Country ?Continent ?Situation) "Not supplied yet."
:Def (And (Country ?Country) (Continent ?Continent) (Sit-Var ?Situation)))
;;; Have-Currency
(Define-Relation Have-Euro (?Continent ?Money ?Situation) "Not supplied yet."
:Def (And (Continent ?Continent)(Money ?Money)(Sit-Var ?Situation)))
;;; Wealthy (Define-Relation Wealthy (?Country ?Money ?Situation) "Not supplied
yet." :Def (And (Contry ?Contry)(Money ?Money)(Sit-Var ?Situation)))
;;; With-Money (Define-Relation With-Money (?Country ?Situation) "Not supplied
yet." :Def (And (Contry ?Contry)(Sit-Var ?Situation)))
;;; Loan-Money-Agent
(Define-Frame Loan-Money-agent :Own-Slots ((Documentation "Not supplied yet.")
(Instance-Of Agent)) :Axioms ((Wealthy Europeanunion Euro [Start])))
;;;Give-Euro-Agent
(Define-Frame Give-Euro-Agent :Own-Slots ((Documentation "Not supplied yet.")
(Instance-Of Agent)) :Axioms ((Have-Currency Europeanunion Euro [Start])))
;;; Accept-EuropeanaUnion-Agent
(Define-Frame Accept-Europeanunion-Agent :Own-Slots ((Documentation "Not
supplied yet.") (Instance-Of Agent)) :Axioms ((Have-Currency Europe Money [(Start)])))

```

;;; Loan-Money

(Define-Axiom Loan-Money "Not supplied yet." := (=> (Indeped-Country ?Country ?Sit1) (And (With-Money ?Country ?Sit2))))

;;; Give-Euro

(Define-Axiom Give-Euro "Not supplied yet." := (=> (Belongs ?Country ?Continent ?Sit1) (And (Indeped-Country ?Country))))

;;; Accept-Europeanunion

(Define-Axiom Accept-Europeanunion "Not supplied yet." := (=> (Located ?Country1 ?Continent1 ?Sit1) (And (Belongs ?Country1 ?Continent1 ?Sit2))))

### PA metaOnt.in

;;; Organisation

(Define-Class Organisation (?X) "Not supplied yet." :Def (And (Thing ?X)))

;;; Country

(Define-Class Country (?X) "Not supplied yet." :Def (And (Thing ?X)))

;;; Agent

(Define-Class Agent (?X) "Not supplied yet." :Def (And (Thing ?X)))

;;; Agent-Needed

(Define-Function Agent-Needed (?Agent-0) :-> ?Value "Not supplied yet." :Def (And (Agent ?Agent-0) (Action ?Value)))

;;; Continent

(Define-Class Continent (?X) "Not supplied yet." :Def (And (Thing ?X)))

;;; Loan-Money-Agent

(Define-Frame Loan-Money-agent :Own-Slots ((Documentation "Not supplied yet.") (Instance-Of Agent)) :Axioms ((Agent-Needed Loan-Money-Agent Loan-Money)(Wealthy Europeanunion Euro [Start])))

;;; Give-Euro-Agent

(Define-Frame Give-Euro-Agent :Own-Slots ((Documentation "Not supplied yet.") (Instance-Of Agent)) :Axioms ((Agent-Needed Give-Euro-Agent Give-Euro)(Have-Currency Europeanunion Euro)))

;;; Accept-EuropeanaUnion-Agent

(Define-Frame Accept-Europeanunion-Agent :Own-Slots ((Documentation "Not supplied yet.") (Instance-Of Agent)) :Axioms ((Agent-Needed Accept-Europeanunion-Agent Accept-Europeanunion)))

```

;;; Indepted-Country
(Define-Individual Indepted-Country (Predicate) "Not supplied yet.")
;;; Belongs
(Define-Individual Belongs (Predicate) "Not supplied yet.")
;;; Share
(Define-Individual Share (Predicate) "Not supplied yet.")
;;; Have-Currency
(Define-Individual Have-Currency (Predicate) "Not supplied yet.")
;;; Wealthy
(Define-Individual Wealthy (Predicate) "Not supplied yet.")
;;; With-Money
(Define-Individual Wealthy (Predicate) "Not supplied yet.")
;;; Loan-Money
(Define-Individual Loan-Money (Action) "Not supplied yet.")
;;; Give-Euro
(Define-Individual Give-Euro (Action) "Not supplied yet.")
;;; Accept-Europeanunion
(Define-Individual Accept-Europeanunion (Action) "Not supplied yet.")

```

### **acceptEuropeanUnionAgent ont.in**

```

;;; Agent
(Define-Class Agent (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Country
(Define-Class Country (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Money
(Define-Class Money (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Continent
(Define-Class Continent (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Sit-Var
(Define-Class Sit-Var (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Organisation
(Define-Class Organisation (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Euro (Define-Individual Euro (Currency) "Not supplied yet.")
;;; Greece

```

```

(Define-Individual Greece (Country) "Not supplied yet.")
;;; Europe
(Define-Individual Europe (Continent) "Not supplied yet.")
;;; EuropeanUnion
(Define-Frame EuropeanUnion :Own-Slots ((Documentation "Not supplied yet.")
(Instance-Of Organisation)) :Axioms ((Have-Currency Europe Money [(Start)]))
;;; Pseudo-Var
(Define-Individual Pseudo-Var (Confirmation-Number) "Not supplied yet.")
;;; Sit-Var
(Define-Class Sit-Var (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Start
(Define-Individual Start (Sit-Var) "Not supplied yet.")
;;; Located
(Define-Relation Located (?Continent ?Country ?Situation) "Not supplied yet."
:Def (And (Continent ?Continent) (Country ?Country) (Sit-Var ?Situation)))
;;; Belongs
(Define-Relation Belongs (?Country ?Continent ?Situation) "Not supplied yet."
:Def (And (Country ?Country)(Continent ?Continent)(Sit-Var ?Situation)))
;;; Have-Currency
(Define-Relation Have-Euro (?Continent ?Money ?Situation) "Not supplied yet."
:Def (And (Continent ?Continent)(Money ?Money)(Sit-Var ?Situation)))
;;; Accept-EuropeanaUnion-Agent
(Define-Frame Accept-Europeanaunion-Agent :Own-Slots ((Documentation "Not
supplied yet.") (Instance-Of Agent)) :Axioms ((Located Europe Greece [Start]))
;;; Accept-Europeanunion
(Define-Axiom Accept-Europeanunion "Not supplied yet." := (= > (Located ?Con-
tinent ?Country ?Sit1) (And (Belongs ?Country ?Continent ?Sit2))))

```

### **acceptEuropeanUnionAgent metaOnt.in**

```

;;; Action
(Define-Class Action (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Loan-Money-Agent
(Define-Frame Accept-Europeanunion-Agnet :Own-Slots ((Documentation "Not
supplied yet.") (Instance-Of Agent)) :Axioms ((Tasks-I-Perform Accept-Europeanunion)(Ask

```

```

Located)(Protect-Predicate Located Predicate-All Low-Protection)))
;;; Belongs
(Define-Individual Belongs (Predicate) "Not supplied yet.")
;;; Located
(Define-Individual Located (Predicate) "Not supplied yet.")
;;; Share (Define-Individual Share (Predicate) "Not supplied yet.")
;;; Have-Currency (Define-Individual Have-Currency (Predicate) "Not supplied
yet.
;;; Accept-Europeanunion
(Define-Individual Accept-Europeanunion (Action) "Not supplied yet.
;;; PROTECTED
;;; Predicate-Option
(Define-Class Predicate-Option (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Argument-Option
(Define-Class Argument-Option (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Protection-Level
(Define-Class Protection-Level (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Protect-Predicate
(Define-Function Protect-Predicate (?Relation ?Value) :-> ?Level "Not supplied
yet." :Def (And (Relation ?Relation) (Predicate-Option ?Value) (Protection-Level ?Level)))
;;; Protect-Argument
(Define-Function Protect-Argument (?Relation ?Argument ?Value) :-> ?Level "Not
supplied yet." :Def (And (Relation ?Relation) (Argument ?Argument) (Argument-Option
?Value) (Protection-Level ?Level)))
;;; Predicate-All
(Define-Individual Predicate-All (Predicate-Option) "Not supplied yet.")
;;; Predicate-Arity
(Define-Individual Predicate-Arity (Predicate-Option) "Not supplied yet.")
;;; Argument-All
(Define-Individual Argument-All (Argument-Option) "Not supplied yet.")
;;; Argument-Value
(Define-Individual Argument-Value (Argument-Option) "Not supplied yet.")
;;; Argument-Class
(Define-Individual Argument-Class (Argument-Option) "Not supplied yet.")
;;; High-Protection

```

```
(Define-Individual High-Protection (Protection-Level) "Not supplied yet.")
;;; Low-Protection
(Define-Individual Low-Protection (Protection-Level) "Not supplied yet.")
;;; ~ PROTECTED
```

### **giveEuroAgent ont.in**

```
;;; Agent
(Define-Class Agent (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Country
(Define-Class Country (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Money
(Define-Class Money (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Continent
(Define-Class Continent (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Sit-Var
(Define-Class Sit-Var (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Organisation
(Define-Class Organisation (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Euro
(Define-Individual Euro (Currency) "Not supplied yet.")
;;; Greece
(Define-Individual Greece (Country) "Not supplied yet.")
;;; Europe
(Define-Individual Europe (Continent) "Not supplied yet.")
;;; EuropeanUnion
(Define-Frame EuropeanUnion :Own-Slots ((Documentation "Not supplied yet.")
(Instance-Of Organisation)) :Axioms ((Have-Currency Europe Money [(Start)])))
;;; Pseudo-Var
(Define-Individual Pseudo-Var (Confirmation-Number) "Not supplied yet.")
;;; Sit-Var
(Define-Class Sit-Var (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Start
(Define-Individual Start (Sit-Var) "Not supplied yet.")
;;; Indepted-Country
```

```

(Define-Relation Indepted-Country (?Country ?Situation) "Not supplied yet." :Def
(And (Country ?Country)(Sit-Var ?Situation)))
;;;Belongs
(Define-Relation Belongs (?Country ?Continent ?Situation) "Not supplied yet."
:Def (And (Country ?Country)(Continent ?Continent)(Sit-Var ?Situation)))
;;; Have-Currency
(Define-Relation Have-Euro (?Continent ?Money ?Situation) "Not supplied yet."
:Def (And (Continent ?Continent)(Money ?Money)(Sit-Var ?Situation)))
;;;Give-Euro-Agent
(Define-Frame Give-Euro-Agent :Own-Slots ((Documentation "Not supplied yet.")
(Instance-Of Agent)) :Axioms ((Have-Currency Europeanunion Euro [Start])))
;;;Give-Euro
(Define-Axiom Give-Euro "Not supplied yet." := (=> (Belongs ?Country ?Conti-
nent ?Sit1) (And (Indepted-Country ?Country))))

```

### **giveEuroAgent metaOnt.in**

```

;;; Inform
(Define-Class Inform (?X) "Not supplied yet." :Def (And (Predicate ?X)))
;;;Action
(Define-Class Action (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Give-Euro-Agnet
(Define-Frame Give-Euro-Agnet :Own-Slots ((Documentation "Not supplied yet.")
(Instance-Of Agent)) :Axioms ((Tasks-I-Perform Give-Euro) (Ask Belongs)))
;;; Belongs
(Define-Individual Belongs (Predicate) "Not supplied yet.")
;;; Indepted-Country
(Define-Individual Indepted-Country (Predicate) "Not supplied yet.")
;;; Have-Currency
(Define-Individual Have-Currency (Predicate) "Not supplied yet.")
;;;Give-Euro
(Define-Individual Give-Euro (Action) "Not supplied yet."

```

### **loanMoneyAgent ont.in**

```

;;; Agent

```

```

(Define-Class Agent (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Country
(Define-Class Country (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Money
(Define-Class Money (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Continent
(Define-Class Continent (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Sit-Var
(Define-Class Sit-Var (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Organisation
(Define-Class Organisation (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Euro
(Define-Individual Euro (Currency) "Not supplied yet.")
;;; Greece
(Define-Individual Greece (Country) "Not supplied yet.")
;;; Europe
(Define-Individual Europe (Continent) "Not supplied yet.")
;;; EuropeanUnion
(Define-Frame EuropeanUnion :Own-Slots ((Documentation "Not supplied yet.")
(Instance-Of Organisation)) :Axioms ((Have-Currency Europe Money [(Start)]))
;;; Pseudo-Var
(Define-Individual Pseudo-Var (Confirmation-Number) "Not supplied yet.")
;;; Sit-Var
(Define-Class Sit-Var (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Start (Define-Individual Start (Sit-Var) "Not supplied yet.")
;;; Indepted-Country
(Define-Relation Indepted-Country (?Country ?Situation) "Not supplied yet." :Def
(And (Country ?Country)(Sit-Var ?Situation)))
;;; Located
(Define-Relation Located (?Country ?Continent ?Situation) "Not supplied yet."
:Def (And (Country ?Country)(Continent ?Continent) (Sit-Var ?Situation)))
;;; Wealthy
(Define-Relation Wealthy (?Country ?Money ?Situation) "Not supplied yet." :Def
(And (Contry ?Contry)(Money ?Money)(Sit-Var ?Situation)))
;;; With-Money

```

```

(Define-Relation With-Money (?Country ?Money ?Situation) "Not supplied yet."
:Def (And (Contry ?Contry)(Money ?Money)(Sit-Var ?Situation)))
;;; Loan-Money-Agent
(Define-Frame Loan-Money-agent :Own-Slots ((Documentation "Not supplied yet.")
(Instance-Of Agent)) :Axioms ((Wealthy EuropeanUnion Euro [Start])))
;;; Loan-Money
(Define-Axiom Loan-Money "Not supplied yet." := (=> (Indepted-Country ?Country ?Sit1) (And (With-Money ?Country ?Sit2))))

```

### **loanMoneyAgent metaOnt.in**

```

;;; Inform
(Define-Class Inform (?X) "Not supplied yet." :Def (And (Predicate ?X)))
;;; Action
(Define-Class Action (?X) "Not supplied yet." :Def (And (Thing ?X)))
;;; Loan-Money-Agent
(Define-Frame Loan-Money-agent :Own-Slots ((Documentation "Not supplied yet.")
(Instance-Of Agent)) :Axioms ((Tasks-I-Perform Loan-Money) (Ask Indepted-Country)))
;;; Indepted-Country
(Define-Individual Indepted-Country (Predicate) "Not supplied yet.")
;;; Located
(Define-Individual Located (Predicate) "Not supplied yet.")
;;; Share
(Define-Individual Share (Predicate) "Not supplied yet.")
;;; Wealthy
(Define-Individual Wealthy (Predicate) "Not supplied yet.")
;;; With-Money
(Define-Individual Wealthy (Predicate) "Not supplied yet.")
;;; Loan-Money
(Define-Individual Loan-Money (Action) "Not supplied yet.")

```

# Appendix B

## Protocol evaluation

In what follows we present the NuSMV code for the validation of our negotiation protocol.

```
MODULE main
VAR
state : { start, callSent, callReceived, callRefused, proposalSent, proposalAccepted,
participantLeftNegotiation, proposalRefused, end };
action : { cfp, receiveCall, refuseCall, propose, acceptProposal, failure, refuseProposal, initiatorDoesRepair, participantDoesRepair };
ASSIGN
init(state) := start;
next(state) := case
state = start & action= cfp : callSent;
state = callSent & action = receiveCall : callReceived ;
state = receiveCall : { callRefused , proposalSent };
state = receiveCall & action = propose : proposalSent;
state = receiveCall & action = refuseCall : callRefused;
state = callRefused & action = failure : participantLeftNegotiation;
state= participantLeftNegotiation & action = initiatorDoesRepair : end;
state = proposalSent : { proposalAccepted , proposalRefused };
state = proposalSent & action = acceptProposal: proposalAccepted;
state = proposalSent & action = refuseProposal: proposalRefused;
state = proposalAccepted & action = participantDoesRepair : end;
state = proposalRefused & action = initiatorDoesRepair : end;
TRUE : state;
```

esac;

FAIRNESS (state = start) ;

FAIRNESS (state= callSent);

SPEC AG(EF(state=end));

LTLSPEC G(F(state=callReceived → F(action=refuseCall—action=propose)))

LTLSPEC G(F(state=proposalSent → F(action=refuseProposal—action=acceptProposal)))

# Appendix C

## Console outputs

**Negotiation outputs for economy scenario** The negotiation takes place between the PA and *acceptEuropeanUnionAgent*. The PA's representation is *located(greece, europe)* and the *acceptEuropeanUnionAgent*'s representation is *located(europe,greece)*. The negotiation object is the *switch arguments repair*.

**PA:**The inverse repair which I am going to negotiate is *switchArgs* I am informing the SPA: *acceptEuropeanunionAgent* about the repair the ORS suggested so he can check its utility. The repair I am sending him is *switchArgs* The cause of the failure is because of the "Located"

**acceptEuropeanUnionAgent:** \* I was asked to repair my ontology. The repair is: *switchArgs*. Information I received: *Located*. I must check if the repair can be performed. Protection used: (*located,predicateAll,lowProtection*). For the repair *switchArgs* my Utility is 1. I sent my utility for this repair: 1

**PA:** I received a response for SPA Utility, which is 1. I want to consult an expert agent. \*\* I just asked agent *acceptEuropeanunionAgent* if it also wants to get the expert agent advice.

**acceptEuropeanUnionAgent:** I was asked if I want an expert advice. And I replied no.

**PA:** It replied no . We did not agree to consult an expert agent.

**acceptEuropeanUnionAgent:** We did not agree to ask for the advice.

**PA:** My utility for this negotiation round is: 2. I am assigned with the role of the initiator, therefore I am starting the negotiation. I am sending to the SPA the negotiation ontology path: *~/trunk/andriana/new-ors/ORS/NegotiationOntology.pl*

**acceptEuropeanUnionAgent:** I received the negotiation ontology path: *~/trunk/andriana/new-ors/ORS/NegotiationOntology.pl* The negotiation ontology has been consulted. I am

assigned with the role of the participant

**PA:** I am sending a cfp query to the SPA,acceptEuropeanunionAgent for the repair switchArgs

**acceptEuropeanUnionAgent:** I sent a receiveCall for the negotiation objectswitchArgs.

**PA:** I received a receiveCall for the negotiation object switchArgs. I need more details to continue the negotiation.

**acceptEuropeanUnionAgent:** I sent a propose for the negotiation objectswitchArgs.

**PA:** I received a propose for the negotiation object switchArgs. I sent a refuseProposal for the negotiation object switchArgs.

**acceptEuropeanUnionAgent:** I received a refuseProposal for the negotiation objectswitchArgs. I need more details to continue the negotiation.

**PA:** I sent a initiatorDoesRepair for the negotiation object switchArgs.

**acceptEuropeanUnionAgent:** The negotiation is over. The outcome is : initiator-DoesRepair

**PA:** The negotiation is over. I am performing the repair switchArgs in Located.

# Bibliography

- [Alani et al., 2006] Alani, H., Harris, S., and O'Neil, B. (2006). Wining ontologies based on application use.
- [Austin, 1962] Austin, J. L. (1962). *How to Do things With Words*. Oxford University Press: Oxford, England.
- [Avery and Yearwood, 2002] Avery, J. and Yearwood, J. (2002). A foundation for strange agent negotiation. Technical Report WS-02-09, AAI.
- [Bailin and Truszkowski, 2001] Bailin, S. C. and Truszkowski, W. (2001). Ontology negotiation between scientific archives. In *SSDBM'01*, pages 245–250.
- [Bomersbach, 2011] Bomersbach, A. (2011). Negotiating mismatches. Master's thesis, University of Edinburgh, School of Informatics, Edinburgh, UK.
- [Bundy and McNeill, 2006] Bundy, A. and McNeill, F. (2006). Representation as a fluent: An ai challenge for the next half century.
- [Chandrasekaran et al., 1999] Chandrasekaran, B., Josephson, J. R., and Benjamins, V. R. (1999). What are ontologies, and why do we need them? *IEEE Intelligent Systems*, 14:20–26.
- [Cimatti et al., 2000] Cimatti, A., Clarke, E., Giunchiglia, F., and Roveri, M. (2000). Nusmv: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2:2000.
- [Farquhar et al., 1996] Farquhar, A., Fikes, R., J., and Rice (1996). The ontolingua server: A tool for collaborative ontology construction. *Technical Report, Stanford KSL*.

- [Ferber, 1999] Ferber, J. (1999). *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.
- [Finin et al., 1994] Finin, T., McKay, D., Fritzson, R., and McEntire, R. (1994). Kqml: An information and knowledge exchange protocol. In *Knowledge Building and Knowledge Sharing*.
- [Finkelstein et al., 1993] Finkelstein, A., Gabbay, D. M., A. Hunter, A., Kramer, J., and Nuseiveh, B. (1993). Inconsistency handling in multi-perspective specifications. In *European Software Engineering Conference*, page 8499.
- [Fox and Long, 2003] Fox, M. and Long, D. (2003). An extension to pddl for expressing temporal planning domains.
- [Genesereth, 1991] Genesereth, M. R. (1991). Knowledge interchange format. In *Proceedings of the Second International Conference in Principles of Knowledge Representation and Reasoning*, pages 599–600.
- [Ghallab et al., 2004] Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning: Theory and Practice*.
- [Gruber, 1992] Gruber, T. R. (1992). Ontolingua: A mechanism to support portable ontologies. *Technical Report KSL*.
- [Gruber, 1993] Gruber, T. R. (1993). A translation approach to portable ontology specification. *Knowledge Acquisition*, 5.
- [Haase and Stojanovic, 2005] Haase, P. and Stojanovic, L. (2005). Consistent evolution of owl ontologies. *Proceedings of ESWC*, page 182197.
- [Hayes, 1985] Hayes, P. J. (1985). The second naive physics manifesto. *Theories of the Common-Sense World*.
- [Hoare, 1978] Hoare, C. A. R. (1978). Communicating sequential processes. *Communications of the ACM*, 21:666,677.
- [Jennings et al., 2001] Jennings, N. R., Faratin, P., Lomuscio, A. R., Parsons, S., Sierra, C., and Wooldridge, M. (2001). Automated negotiation: Prospects, methods and challenges. *International Journal of Group Decision and Negotiation*, 10(2):199–215.

- [Klein, 2004] Klein, M. (2004). *Change Management for Distributed Ontologies*. PhD thesis, Vrije Universiteit in Amsterdam.
- [Lomuscio et al., 2000] Lomuscio, A. R., Wooldridge, M., and Jennings, N. R. (2000). A classification scheme for negotiation in electronic commerce.
- [Lopes et al., 2008] Lopes, F., Wooldridge, M., and Novais, A. Q. (2008). Negotiation among autonomous computational agents: principles, analysis and challenges. *Artif. Intell. Rev.*, 29:1–44.
- [McNeill and Bundy, 2007a] McNeill, F. and Bundy, A. (2007a). Dynamic, automatic, first-order ontology repair by diagnosis of failed plan execution. *IJSWIS (International Journal on Semantic Web and Information Systems) special issue on Ontology Matching*, 3:1–35.
- [McNeill and Bundy, 2007b] McNeill, F. and Bundy, A. (2007b). Dynamic, automatic, first-order ontology repair by diagnosis of failed plan execution. *International Journal on Semantic Web and Information Systems, Special Issue on Ontology Matching*, 3:1–35.
- [McNeill et al., 2004] McNeill, F., Bundy, A., and Walton, C. (2004). Diagnosing and repairing ontological mismatches. In *Proceedings of the second starting AI Researchers' symposium*, Valencia, Spain.
- [McNeill et al., 2005] McNeill, F., Bundy, A., and Walton, C. (2005). Planning from rich ontologies through translation between representations.
- [Milner, 1989] Milner, R. (1989). *Communication and Concurrency*. Prentice Hall.
- [Moor, 2005] Moor, A. D. (2005). Ontology-guided meaning negotiation in communities of practice. In *2nd ICCTe, Milano*.
- [Novacek et al., 2007] Novacek, V., Laera, L., and Handschuh, S. (2007). Semi-automatic integration of learned ontologies into a collaborative framework.
- [Noy et al., 2006] Noy, N. F., Chugh, A., Liu, W., and Musen, M. A. (2006). A framework for ontology evolution in collaborative environments. page 544558.
- [Schoop et al., 2003] Schoop, M., Jertila, Aida, and List, T. (2003). Negoisst: a negotiation support system for electronic business-to-business negotiations in e-commerce. *Data Knowl. Eng.*, 47:371–401.

- [Searle, 1969] Searle, J. (1969). *Speech Acts*. Cambridge University Press.
- [Sicstus, 2005] Sicstus (2005). <http://www.sics.se/sicstus/docs/latest/html/sicstus/>. Master's thesis.
- [Smith, 1980] Smith, R. G. (1980). The contract net protocol: high level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29:1104–1113.
- [Stojanovic et al., 2002] Stojanovic, L., Maedche, A., Motik, B., and Stojanovic, N. (2002). User-driven ontology evolution management.
- [Strobel, 2002] Strobel, M. (2002). An xml schema representation for the communication design of electronic negotiations. *Computer Networks*, 39:661–680.
- [Supnithi et al., 1999] Supnithi, T., Inaba, A., Ikeda, M., Toyoda, J. I., and Mizoguchi, R. (1999). Learning goal ontology supported by learning theories for opportunistic group formation. *In AIED*.
- [Tamma et al., 2002] Tamma, V., Wooldridge, M., and I. Dickinson (2002). An ontology for automated negotiation. *Proceeding of the AMEC, Bologna*, pages 219–237.
- [Trojahn et al., 2006] Trojahn, C., Moraes, M., Quaresma, P., and Vieira, R. (2006). A negotiation model for ontology mapping. pages 762–768.
- [W3C, 2005] W3C (2005).
- [Wurman et al., 2001] Wurman, P. R., Wellman, M. P., and Walsh, W. E. (2001). A parametrization of the auction design space. *Games and Economic Behavior*, 35:304–338.
- [Zhang et al., 2005] Zhang, X., Lesser, V., and Podorozhny, R. (2005). Multi-dimensional, multistep negotiation. *Autonomous Agents and Multi-Agent Systems*, 10:5–40.
- [Zlotkin and Rosenschein, 1993] Zlotkin, G. and Rosenschein, J. S. (1993). A domain theory for task oriented negotiation. *In Proceedings of the 13th international joint conference on Artificial intelligence - Volume 1*, pages 416–422.