Specifying and Verifying Communities of Web Services using Argumentative Agents

Wei Wan

A Thesis

in

The Concordia Institute

for

Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science (Quality Systems Engineering) at
Concordia University
Montreal, Quebec, Canada

August 2008

Your file Votre référence
ISBN: 978-0-494-45352-0
Our file Notre référence
ISBN: 978-0-494-45352-0

ABSTRACT

Specifying and Verifying Communities of Web Services using Argumentative Agents

Wei Wan

This thesis includes two main contributions: the first one is specifying the use of argumentative agents in the design and development of communities of Web services; the second is using a formal technique to verify communication protocols against given properties for these communities.

Web services that provide a similar functionality are gathered into a single community, independently of their origins, locations, and ways of doing. Associating Web services with argumentative agents that are able to persuade and negotiate with others organizes these Web services in a better way so that they can achieve the goals they set in an efficient way. A community is led by a master component, which is responsible among others for attracting new Web services to the community, retaining existing Web services in the community, and identifying the Web services in the community that will participate in composite scenarios. Besides FIPA-ACL, argumentative dialogue games are also used for agent interaction.

In this thesis, we use tableau-based model checking algorithm to verify our argumentative agent-base community of Web services negotiation protocol. This algorithm aims at verifying systems designed as a set of autonomous interacting agents. We provide the soundness, completeness, termination and complexity results.

We also simulate our specification with Jadex BDI programming language and implement our verification with a modified and enhanced version of CWB-NC model checker.

Keywords: Multi-agent systems, BDI agent architecture, model checking, agent oriented programming, FIPA-ACL, dialogue game, agent-based negotiation protocol, Jadex, CWB-NC.

ACKNOWLEDGMENTS

First of all, I would like to thank my supervisor Dr. Jamal Bentahar for his expert guidance, constant encouragement and enduring patience during my Master studies. I am grateful for the opportunity to study at CIISE, which provided an excellent environment in which to cross-fertilize research ideas.

Additionally, I am very grateful to Jihad Labban and Khaled Ghoneim for their comments and suggestions. I would like to thank all the people in my lab for discussions on Multi-agent systems, agent-based Web services and programming. I would also like to thank all the faculty and staff at the CIISE for their assistance during my master's course-work. As for my fellow graduate students who offered great help during my study, I would like to thank them all.

My student life would not have been so much fun without my friends (in alphabetical order): Lei Chen, Xudong Duan, Tania Islam, Tao Long, and Nan Yang. I would like to thank them as well.

Finally, I would like to thank my family, especially my beloved husband Mark for his enduring support and endless love.

Wei Wan

July, 2008

Table of Contents

List of Figures

LIST OF TABLES

Chapter 1. Introduction

In this chapter, we explain what has initiated our interest into the argumentative agent-based communities of Web services and we identify some related technologies. We also specify research problems under consideration, describe our contributions, and present the structure of the thesis.

1.1 Context of Research

This thesis is about designing, developing, and verifying communities of Web services using argumentative agents. We mainly focus on three multi-agent systems (MAS) areas: communication in multi-agent systems, verification of communicating agent-based systems, and Web services interaction using multi-agent systems.

Multi-agent systems are software systems where a set of intelligent agents interact with each other. Agents are considered to be autonomous entities with social ability, reactivity, and pro-activity properties. Rationality is another property in reasoning agents. Their behaviors can be either cooperative or self-interested. In MAS, communication is one of the major topics of research. Multiple agents communicate to coordinate with each other in order to solve problems together. Like human's communication, agent communication involves philosophy of language, social psychology, logics, and mathematics, and integrates other disciplines. When agents want to negotiate in order to solve conflicts of interest, they need not only to exchange single messages, but also to think about the reason behind conflicts and try to remove these conflicts. In other words, agents need to participate in complex conversations with other agents.

Agent Communication Language (ACL) is used to exchange information among the agents. In the early 1990s, the DARPA knowledge sharing effort (KSE) began to develop the

Knowledge Query and Manipulation Language (KQML), which became the first standardized ACL [24]. Five years later, in 1995, the Foundation for Intelligent Physical Agents (FIPA[1]) started developing standards for agent systems, including FIPA-ACL. This language is considered as the centerpiece of agent technology. KQML and FIPA-ACL are both based on *speech act theory* and their messages are considered as communicative acts whose objective is to perform some action by virtue of being sent. FIPA-ACL proposed a set of communication protocols that agents can follow. These protocols usually describe the sequence of messages that agents can exchange for particular applications. However, they are sometimes too rigid to be used by autonomous agents in their negotiation or persuasion conversations. This is because they are specified so that an agent must follow them from beginning to end without questioning about them. Dialogue games are introduced by several researchers to solve this problem [7][29]. Dialogue games are abstract structures that can be composed to reflect the whole dialogue. They are interactions among two or more players with serial predefined rules.

Since verification solves problems related to the satisfaction of design requirements, it plays a crucial role in any development process to avoid unwanted behaviours, especially in communicating multi-agent systems, which are innately more complex than "traditional" systems. This is because multi-agent systems are employed to capture high level properties of large, autonomous systems. In this setting, testing is one of the most common verification techniques. Testing is performed by running a number of test cases and by checking that the required properties hold in all tested runs. However, the main problem of testing is that enumerating all possible cases is generally hard and even impossible sometimes. Hence, our research will investigate techniques using formal verification, which is a class of logic-based techniques. Logic-base techniques include theorem proving and model checking. Compared to theorem proving, which is a semi-automatic approach, model checking is an algorithmic-based and

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

automatic technique. Consequently, our research focuses on model checking, mainly a new model checking algorithm for communicating agents.

Our research is applied to Web services that are changing the way industry does business in the world. We are currently witnessing the rapid development and maturation of emerging interrelated standards that are defining the Web services infrastructure along with a number of development tools. The basic standards for Web services, such as WSDL (Web Services Description Language), UDDI (Universal Description, Discovery, and Integration), SOAP (Simple Object Access Protocol), and XML (eXtensible Markup Language) are made for describing, advertising, discovering and binding Web services in a decentralized, distributed service-oriented environment. The shortage of these standards is that they do not deal with the dynamic composition of existing services [42]. Web services compositions, for example BPEL (Business Process Execution Language), and Web services interactions, such as agent-based communities of Web services, are introduced to solve this problem. Our research is on agent-based communities of Web services using dynamic interaction protocols.

1.2 Motivations

In order to facilitate agile business and to support dynamic partnership formation, information systems are designed to support interoperability. In particular, interoperation between agent systems and Web services is more interesting because of the benefits that can be achieved from both technologies to accomplish complex goals. Both technologies have pursued providing dynamic, open and oriented architectures. But the way of designing and deploying systems using these two technologies are very different. Web services use service composition and service interaction to achieve dynamics, whereas agents can know the environment and react according to the environmental change. Commuinties of Web services are the collection of Web services with same or simular functionalities. Although Web servcies are intensively invertigated, the research

community has not addressed properly the community-related issues yet, for example, how to initiate and specify a community of Web services, and how to manage Web services residing in a community. Web services do not have automous property, therefore, the decision of joining a community is usually not optimized, and the selection of one Web service out of many in a community to participate in a composite scenario is generally random. Our first motivation is to integrate the multi-agent systems and Web services when designing and managing communities of Web services so that these operations will be optimized. With agent's automous and reactivy properties, a community is able to be managed efficiently.

Another motivation is to allow these agent-based Web services to use their autonomous and reasoning features through an argumentation system helping them to negotiate situations within the communication protocols that we provide. Empowering Web services with argumentation capabilities comes with its own challenges. Associating Web services with argumentative agents that are able to persuade and negotiate with others organizes these Web services in a better way so they can achieve the goals they set in an efficient way. Unlike FIPA-ACL protocols, which only describe the sequence of allowed actions without any reasoning, our protocols are designed to be more flexible and autonomous. Agents have more selectable actions and play moves in turn by performing utterances according to a predefined set of logical rules. After communication, agents should get enough information and perform actions to achieve their goals.

Providing efficient formal verification for our integrated model is our third motivation. The verification we need should verify both temporal and action formulae. The aim is to develop a model checking algorithm to verify interacting agent-based Web services. A specific logic for our communication protocols is then needed.

1.3 Research Questions

The overall research questions are the following:

1. How do agent-based communities of Web services participate in flexible conversations (persuasions, argumentative negotiations, deliberations, etc.)? How do we specify and represent communities of Web services in compliance with existing standards/specifications?
2. What kind of architecture do we use? Which platform do we select for applications?
3. How do we verify the communities of Web services? What kinds of algorithms can be used to verify our protocols and what is the computational complexity of these protocols?

1.4 Proposed Solutions and Contributions

In order to answer the proposed research questions, we introduce dialogue game with FIPA-ACL language to make agents communication more pliable. We specify the communication protocols for our community of Web services to make these Web services able to negotiate. BDI agent architecture is adopted and FIPA-ACL compliant agent platform Jadex is used as an application platform. Therefore, our agent-based Web services are able to hold a conversation based on their own knowledge and beliefs to make the decision and run desired plans. For verification, we apply tableau-based model checking technique and extend CTL* to CTL*^{CA} for communicative agents.

The main concern of the thesis is the applicability study of the emerging notion of communities for Web services. Indeed, the contributions of this work are the following:

1. Specification of the communities of Web services using argumentative agent.
2. Design of flexible communication protocols for agent-based Web services.

3. Simulation of the specified communities and interaction protocols using Jadex (a java-based BDI reasoning engine).
4. Specification and implementation of a model checking algorithm to verify the specified communities.

1.5 Outline

This thesis is divided into 6 chapters and 2 appendices. Chapter 2 and Chapter 3 are about the state of the art. Chapter 2 introduces multi-agent systems and agent-oriented programming. Chapter 3 presents an overview of Web services standards and interactions. Chapter 4 and Charter 5 consist of our main contributions. Chapter 4 includes the specification of communities of Web services and its implementation. Chapter 5 addresses the verification of our model for communities of Web services. Finally, Chapter 6 concludes the thesis by summarizing our contributions and identifying directions for future work.

Chapter 2. Multi-Agent Systems and Agent Oriented Programming

This chapter presents and discusses a literature review about multi-agent systems and agent oriented programming. First, Sections 2.1, 2.2 and 2.3 briefly introduce the notions of intelligent agents, multi-agent systems, agent architectures, and communication among agents. Then, Section 2.4 presents some agent-based software development, especially Java-based agent programming.

2.1 Autonomous Agents and Multi-Agent Systems

2.1.1 Intelligent Agents

An agent is defined by Wooldridge [50] as a computer system that is capable of independent action on behalf of its user or owner, situated in certain environment, and capable of autonomous action in this environment in order to meet its design objectives. Agents embrace stronger notion of autonomy than objects in object oriented paradigm, and in particular, they decide for themselves whether or not to perform an action requested from another agent. In general, an agent can perceive reason about, and initiate activities in its environment. It has control over its internal state and its own behavior. It experiences environment through sensors and acts through effectors. An agent also can communicate with other agents, even human beings, using an agent communication language, which we will discuss in Section 2.3.1.

An intelligent agent is an agent with reactive, proactive, and social properties.

Reactive: A reactive system is one that maintains an ongoing interaction with its environments, and responds to changes that occur in it.

Proactive: Generally, proactive agents behaviour are goal directed. An intelligent agent can generate and attempt to achieve goals by taking the initiative in order to satisfy its design objectives.

Social: Agents social ability is the ability to interact with other agents in the system (and possibly humans) through a given agent communication language, and cooperate and compete with others.

Intelligent agents act rationally and predictably. They do things to satisfy certain objectives and pursue their goals rationally. A detailed analysis of "rationality" can be found in the Bratman's [14] work. This analysis opens the way to the Beliefs-Desires-Intentions (BDI) model for software agents [39].

2.1.2 Multi-Agent Systems

A multi-agent system (MAS) is a software system including of a number of intelligent agents, which interact with one-another [50]. A multi-agent system is able to reach goals that are difficult to achieve by an individual agent. Typically, agents exchange messages through specific agent communication languages. In the most general case, agents are acting on behalf of users with different goals and motivations. In order to successful interact, they will require the ability to *cooperate, coordinate,* and possibly *negotiate* with others.

Multi-agent systems are interdisciplinary: the field of MAS is influenced and inspired by many other fields: economics, mathematics, logics, philosophy, game theory, ecology, social sciences, etc. This aspect can be both strength and a weakness: this infuses well-founded methodologies into the field; however, there are many different views as to what the field is about.

To design a MAS, we need to implement micro and macro designs: *agent design* and *society design* [50]. During the agent design (micro design), we think about how to build agents capable of independent, autonomous action so that they can successful carry out tasks users delegate to them. More details about this design will be discussed in Section 2.2 through agent

architecture. In society design (macro design), the focus is on the interaction capabilities (cooperation, coordination, and negotiation) in order to successfully carry out the delegated tasks, especially when some conflicts arise in agent goals.

2.2 Agent Architectures

2.2.1 Preliminaries

Agent Architectures are significant when we delve into agents design. They are the foundation of the agent paradigm. In general, agent architecture is a framework that is a blueprint for software agents and intelligent control system, depicting the arrangement of components. Through the analysis of agent architecture, we can not only predict and explain the behavior of an agent system based on investigating its current state and environment, but also get a particular methodology for building real agent systems. According to Maes' [28], agent architecture specifies how "the agent can be decomposed into the construction of a set of component modules and how these modules should be made to interact."

There are many ways to classify the agent architectures. Typically, for single agent architectures, identified three categories are identified by Wooldridge [50]:

1. Deliberative agent architecture (symbolic/logical): an agent contains an explicitly represented, symbolic model of the environment. Furthermore, it makes decisions via symbolic reasoning.
2. Reactive agent architecture: an agent acts by means of stimulus-response rules and does not symbolically represent their environment (explained through agent-oriented programming)
3. Hybrid agent architecture: an agent can act both deliberatively and reactively.

In the beginning, symbolic reasoning agent architecture was dominant. In this architecture, agents have been based on logical or symbolic reasoning to decide about their

actions. There were three types of reasoning agents: symbolic reasoning agents, deductive reasoning agents and practical reasoning agents. *Belief-Desire-Intention* (BDI) architecture is one of the main deliberative agent architectures.

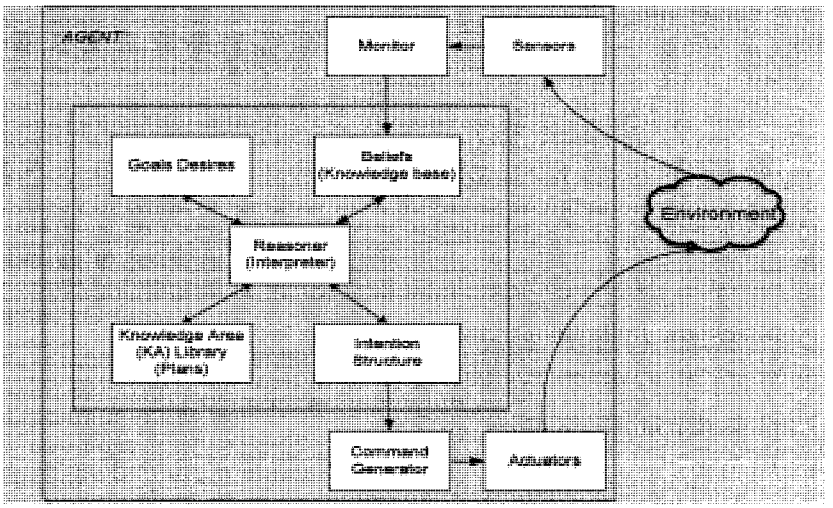

Figure 2.1 The PRS agent architecture

Figure 2.1 is *Procedural Reasoning System* (PRS) architecture which builds its agent systems using BDI architecture. A reasoner, also called an interpreter, is responsible for maintaining beliefs (Knowledge base) about the world state, choosing which goals to attempt to achieve next, and choosing which Knowledge Area (KA) to apply in the current situation. A PRS agent can explore the environment changes through sensors; then, based on the beliefs of the environment, PRS interleaves planning and doing action in the world.

The "reactive agent movement" started in 1985, revealed an era of reaction agent architecture. The agents' behavior has not relied solely on symbol manipulation, but also generated reaction without explicit abstract reasoning. The agents in reactive architecture are capable of maintaining an ongoing interaction with the environment, and responding to changes that occur in it [52]. Rodney Brooks' subsumption architecture [15], which considers agent properties and capabilities along with the environment, is an excellent example of reactive agent architecture.

Since 1990, hybrid architectures that combine the best of reasoning and reactive architectures have come to the forefront. In this architecture, the agent designer can build an agent out of two or more subsystems: one is a deliberative agent, which contains a symbolic world model that develops plans and makes decisions in the way proposed by symbolic agents; another is a reactive one that is capable of reacting to events without complex reasoning.

For space and comprehensibility, we only discuss BDI agent architecture because it provides a foundation for many systems and is considered separately as an abstract architecture in its own right.

2.2.2 BDI Agent Architecture

BDI architecture, which was introduced by Michael Bratman as a philosophical model for describing rational agents [14], contains specific denotation of Beliefs, Desires, and Intentions [39] and addresses how Beliefs, Desires and Intentions are represented, updated and processed. In BDI agent architecture, agents are imbued with particular mental attitudes and choose the appropriate action according to their capability and the internal structures.

- **Beliefs**: Beliefs represent how agents know the surroundings, including themselves and other agents. Beliefs can also include inference rules, allowing forward chaining to lead to new beliefs. In general, this information is stored in a database that usually called a *belief base*. Unlike knowledge, beliefs may be not true.
- **Desires:** Desires (or goals) are things that agents would like to accomplish [41]. They are the motivational state of the agent. Examples of desires might be: *find the best price*, *go to the party* or *become rich*, etc. The difference between goals and desires is that the set of goals must be consistent, while desires could be inconsistent.
- **Intentions:** Intentions are the agents' targets. They represent the deliberative state of the agent: what the agent *has chosen* to do. Intentions are desires to which the agent has to some extent committed (in implemented systems, this means the agent has begun

executing a plan). Plans are sequences of actions that an agent can perform to achieve one or more intentions. Plans may include other plans: a plan to go for a drive may include a plan to find the car keys. In Bratman's model, plans are initially only partially conceived, with details being filled in as they progress.

Typically, the BDI model combines three distinct components [51]: a philosophical component (theory of humans rational action); a software architecture component (used in a number of complex applications); and a logical component (BDI logics), such as LORA (Logic of Rational Agents).

The diagram below from [41] indicates a BDI model execution Cycle. When new information arrives, the agents will update their beliefs and desires (goals). These new beliefs or goals may be able to trigger several actions. However, only an intended action is selected and activated. After this action is performed, intentions are updated and the new beliefs or goals are stored. New cycle is started. More details about DBI programming are presented in Section 2.4.

2.3 Agent Communication and Argumentation-based Agents

2.3.1 Agent Communication Languages

Agent Communication Languages (ACL) are proposed standard languages for communicating agents. Most ACLs are based on the speech-act theory that is expressed by means of standard keywords, known as 'performatives' [47]. There are two main ACLs over the last decade: the Knowledge Query and Manipulation Language (KQML), which was proposed by DARPA Knowledge Sharing Effort (KSE), and the foundation for Intelligent Physical Agents' Agent Communication Language (FIPA-ACL).

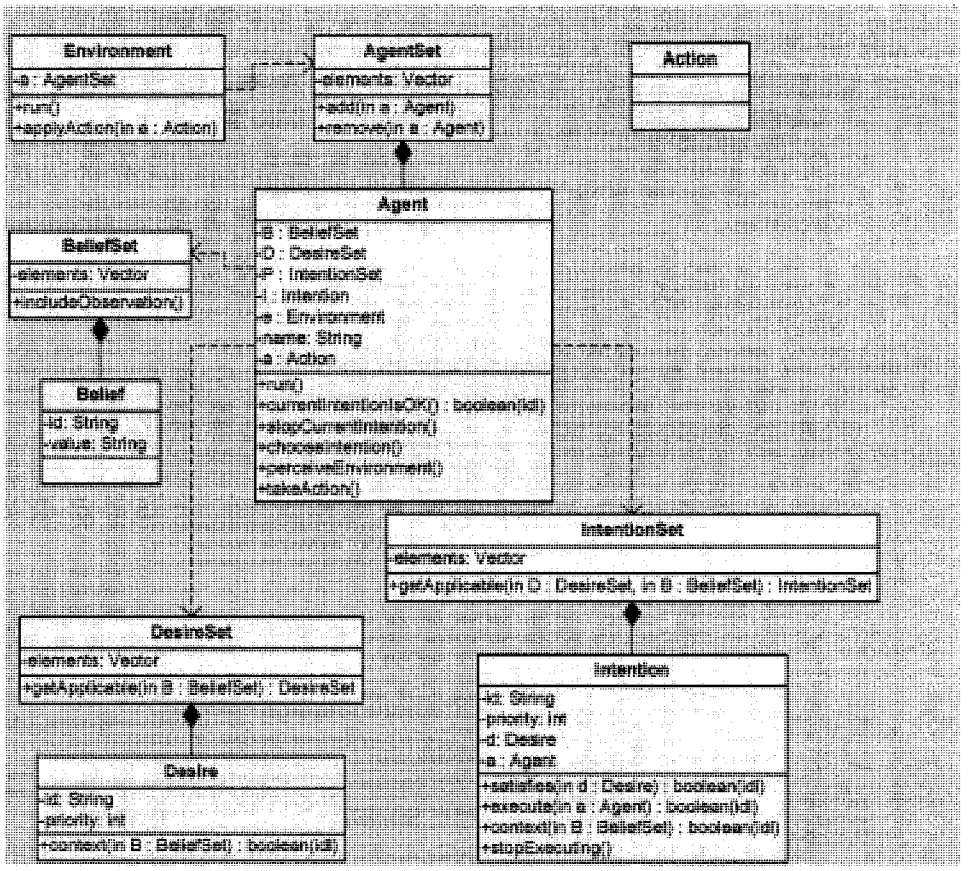

Figure 2.2 Diagram of a Belief-Desire-Intention architecture

2.3.1.1 KQML

KQML is the notion of performative keyword, such as *ask-if, tell, ask-one, etc.* There are roughly 41 performatives that define the intended meaning of a message. An example of KQML, which I picked up from [11], will help to understand better:

```
(ask-one
        :content(PRICE IBM ?price)
        :receiver       stock-server
        :language       LPROLOG
        :ontology       NSE-TICKS)
```

To interpret this KQML message, first, look at the performative: ask-one. This message stipulates that sender wants to ask some information. The *content* explains what kind of information the sender needs: the price of PRICE IBM. The message is sent to *receiver:* stock-

server and is written in a language called LPROLOG. A particular ontology (NYSE-TICKS) defines the terminology used in the message.

KQML was proposed lacking any formal semantics at first. Near the end of the 1990s, Labrou and Finin introduced some semantics into KQML [26] though the semantics were just presented using a pre- and post-condition notation using a logical language containing modalities for belief, knowledge, wanting, and intending [51].

2.3.1.2 FIPA-ACL

Foundation for Intelligent Physical Agents Standards (FIPA) is a nonprofit organization founded in 1996 to provide software standards specifications for interoperability between heterogeneous agents and agent based systems. FIPA has published 25 standard specifications, including a further 14 remaining at the experimental stage, and 3 at the preliminary stage [1]. Specially, FIPA-ACL message structures are defined by FIPA Agent Communication Standards. The components of the communication model are shown in Figure 2.3

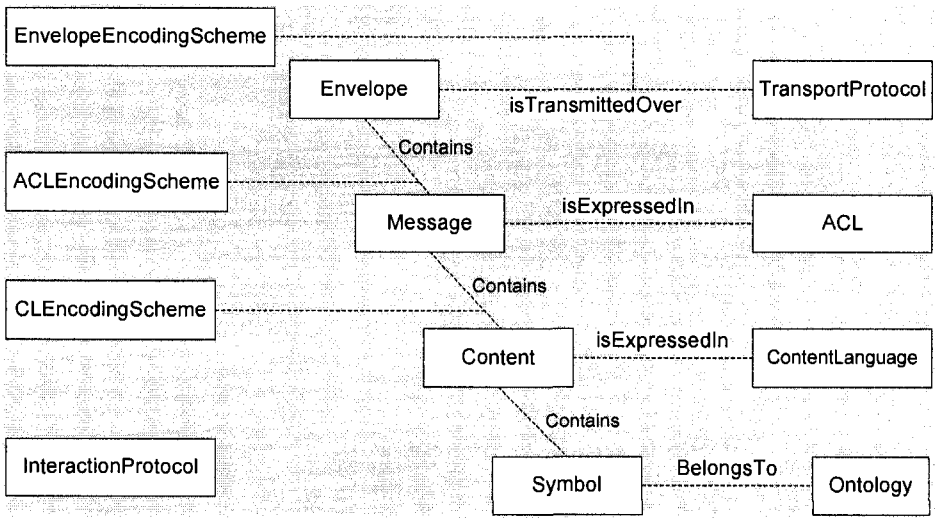

Figure 2.3 FIPA Standard: components of the communication model

- Message Transport Protocol (MTP): this level carries out the physical transfer of messages between Agent Communication Channel (ACC).

- Message Transport Service (MTS): this level is provided by the agent platform to which an agent is attached. The MTS supports the transport of FIPA-ACL messages between agents on any given agent platform and between agents on different agent platforms.

A FIPA-compliant message that is the fundamental form of communication between agents composes four parts: Envelope, Payload, Message, and Content. The general structure of a message is showed below figure 2.4.

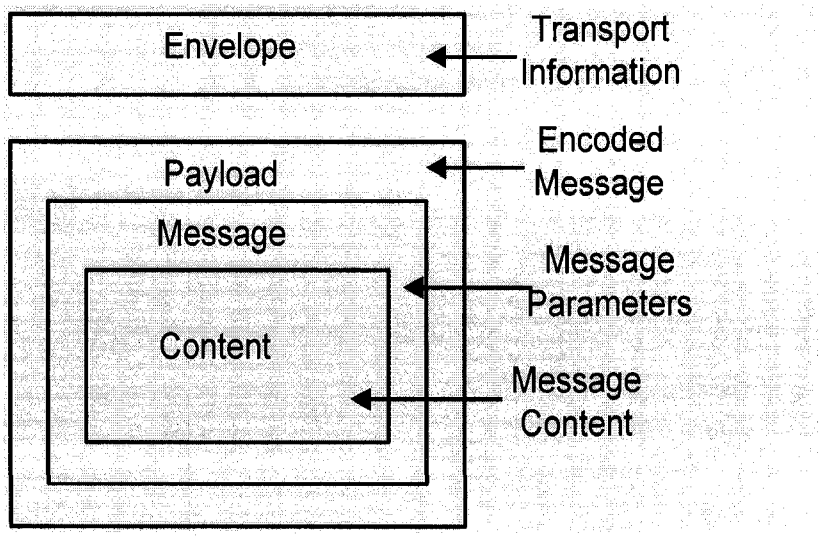

Figure 2.4 FIPA message structure

FIPA-SL (Semantic Language) or FIPA-KIF (Knowledge Interchange Format) is a content language that expresses the content of the message. These content expressions can be some kind of symbols that are grounded by referenced ontology. Message parameters must include key parameters such as performative, sender's and receiver's Agent identification (AID). The messages will be encoded into a payload before they are transmitted and a message transport envelope for the particular protocol in use, like HTTP or IIOP is used.

A FIPA-ACL message contains message parameters that are needed for effective agent communication. The only mandatory parameter in any ACL message is the "*performative.*" Other frequently used parameters also contain *sender*, *receiver* and *content*. The FIPA-ACL message parameters are shown in Table 2.1.

Table 2.1 ACL Message Parameters

Parameter	Description
performative	Type of the communicative act of the message
Sender	Identify of the sender of the message
Receiver	Identify of the intended recipients of the message
reply-to	Which agent to direct subsequent message to within a conversation thread
Content	Content of the message
Language	Language in which the content parameter is expressed
Encoding	Specific encoding of the message content
Ontology	Reference to an ontology to give meaning to symbols in the message content
Protocol	Interaction protocol used to structure a conversation
conversation-id	Unique identify of a conversation thread
reply-with	An expression to be used by a responding agent to identify the message
in-reply-to	Reference to an earlier action to which the message is a reply
reply-by	A time/date indicating by when a reply should be received

There is a simple example of a FIPA-ACL message with a *request* performative:

```
(request
        :sender (:name dominic-agent@whitestein.com:8080)
        :receiver (:name rex-hotel@tcp://hotelrex.com:6600)
        :ontology personal-travel-assistant
        :language FIPA-SL
        :protocol fipa-request
        :content
                (action movenpick-hotel@tcp://movenpick.com:6600
                  (book-hotel   (:arrival 25/11/2000)
                        (:departure 05/12/2000)...))
)
```

2.3.2 Argumentation-based Agents

In [33], Pavlos Moraitis abstracted argumentation definition as the principle interaction of different, potentially conflicting arguments to obtain a consistent conclusion [54]. Like human beings, argumentation-based agents rely on the dialogues or messages being exchanged to attempt to achieve their goals or desires. In this thesis, the argumentation in multi-agent systems

is not only a process by which one agent tries to convince another to accept some beliefs, but also a process by which one agent looks for information from others.

Argumentation-based reasoning is an advanced type of reasoning, more efficient than classical reasoning based on deduction or abduction [10]. Before argumentative agents make decisions based on evaluating arguments, they can deliberate about their beliefs and goals [8]. At the architecture level, argumentative agents are more autonomous than normal deliberative agents: they are BDI agents augmented with additional capabilities [6].

The Argumentative agents can engage in an argumentation based dialogue to negotiate, to persuade, to seek information, to inquire, or to deliberate. In three of five different dialogues, information seeking, inquiry, and deliberation, the sender does not try to change the receivers' beliefs. The sender just seeks answers to some questions or to jointly agree a course of action in specific situation [30]. It either gets information from a receiver (receivers), or tells a receiver (receivers) its decision.

However, in negotiation dialogues, the sender who initiates the dialogue and the receiver who participates in the dialogue actively seek the best deal for both parties involved. The sender in persuasion dialogues wants to persuade another agent to accept the information the sender provided.

There are several argumentation theories and frameworks. In this thesis, I adopt the framework from [6]. An argumentation system essentially comprises three components: **a logical language $\mathcal{L}$, a definition of the *argument concept*,** and **a definition of the *attack relation* between arguments**. The logical language enables argumentative agents to "think" logically: inferring and justifying conclusion with a logic-based reasoning.

Definition of the argument concept: *Let Γ be a knowledge base with no deductive closure. An argument is a pair (H,h) where h is a formula of $\mathcal{L}$, and H is a subset of Γ such that:*

(i)H is consistent, (ii) H h, and (iii) H is minimal, so that no subset of H satisfying both (i) and (ii) exists. H is called the support of the argument and h its conclusion.

Definition of the attack relation: *Let AT be a binary relation between arguments, and (H,h) and (H', h') be two arguments. (H', h') AT (H, h) iff H' ' ¬ h. In other words, an arguments is attacked if and only if there exists an argument that negates its conclusion.*

2.3.3 Negotiation

The definition of negotiation from Webster's dictionary is "discussion, argument, or bargaining with others in search of an agreement". From this definition, negotiation is the process of reaching agreements on matters of common interest. There are two types of negotiation domains [40]: task-oriented domains and worth-oriented domains.

Generally speaking, any negotiation setting includes four different components [50]: negotiation set, protocol, collection of strategies, and a rule.

1. **A negotiation set**: possible proposals that agents can make, such as the value of credit, the price of book, etc.
2. **Protocol**: a particular mechanism manipulates negotiation.
3. **Collection of strategies**, each agent has his own private strategies.
4. **A rule**, which determines when a deal has been struck and what the agreement deal is.

Negotiation dialogues normally do not finish in the first round. An agent, based on his strategies, adjusts his proposal when it is refused another agent's proposal. The dialogue continues until agents either come to an agreement or fail to, stemming from the rule.

2.3.4 Dialogue Games

However, both FIPA-ACL and KQML must follow the protocol sequences. Hence, they are not flexible to suit autonomous agents' negotiation. Using formal dialogue games has been

proposed by McBurney et al.[29] and Bentahar to solve this problem [4]. Dialogue games are abstract structures including communication protocols, which mainly are a pre-defined set of rules that typically define which locutions may or must be uttered in different circumstances. These rules are declarative specifications that manipulated communication between autonomous agents. Therefore dialogue games enable agents to coordinate, even negotiate the dialogical activity.

Dialogue games are interactions between agents, in which each agent moves by performing utterances according the rules, (protocols). The model of a dialogue game we will present is taken from McBurney [29].We suppose that there are certain logical languages, whose well-formed formulae are denoted by the lower-case Roman letters, like *p, q,* represented between the agents. A dialogue game specification then includes five elements as following:

- Commencement Rules: define the circumstances under which a dialogue commences.
- Locutions: indicate what utterances are permitted
- Combination Rules: define the dialogue contexts under whether or not particular locutions are permitted/ obligatory.
- Commitments: define the circumstances under which participants express commitment to a proposition.
- Termination Rules: define the circumstances under which a dialogue ends.

Several dialogue games types have been proposed: information seeking dialogues; inquiry; persuasion dialogues; negotiation dialogues; and deliberations [49]. Our research will focus on negotiation dialogues.

In this thesis, we formalize specific dialogue games for communities of Web service as a set of logical rules about which agents can reason in order to decide which game to play and how to combine games. We propose to specify protocols, which agents must respect them from the beginning to the end, by small dialogue games that can be considered as conversation policies that can be logically put together. The information will be discussed in detail in Chapter 4.

2.4 Agent Oriented Programming

Agent oriented programming has revealed a great potential for developing complex applications using agent technology. There are a number of approaches or methodologies for agent-based programming, such as *Jason* [34], *3APL* [25], *JADE* [1], *Jadex* [38], *and JACK* [16] In this section, we only glance at Java-based FIPA Compliant agent programming Languages, JADE and Jadex that we used to develop our system.

2.4.1 JADE (Java Agent Development framework)

JADE, one of the most widespread agent oriented middleware in use today, provides a FIPA-compliant agent platform and a package to develop Java agents. It is an open-source software that has been under development since 1999 by TILab[2] (Telecom Italia Labs).

The internal architecture of JADE is fully complies with FIPA standard. FIPA compliant agents can exist, operate and be managed. It provides a basic set of functionalities that are regarded as essential for autonomous agents. The architecture of JADE Agent Platform (*AP*) is shown in Figure 2.5 [1].

The first terminology in this architecture is ***Container***, which can contain several agents. A specific container, named ***Main container***, must always be active in a platform, which contains the set of active containers. Other containers register with ***Main container*** as soon as they start. Other important components are ***AMS*** (Agent Management System) and ***DF*** (Directory Facilitator). The ***AMS*** provides the naming and control access service, like a white page, and represents the authority in the platform. The ***DF*** provides a yellow pages service by means of which an agent can find other agents providing the services it requires in order to achieve its goals.

[2] http://jade.tilab.com/

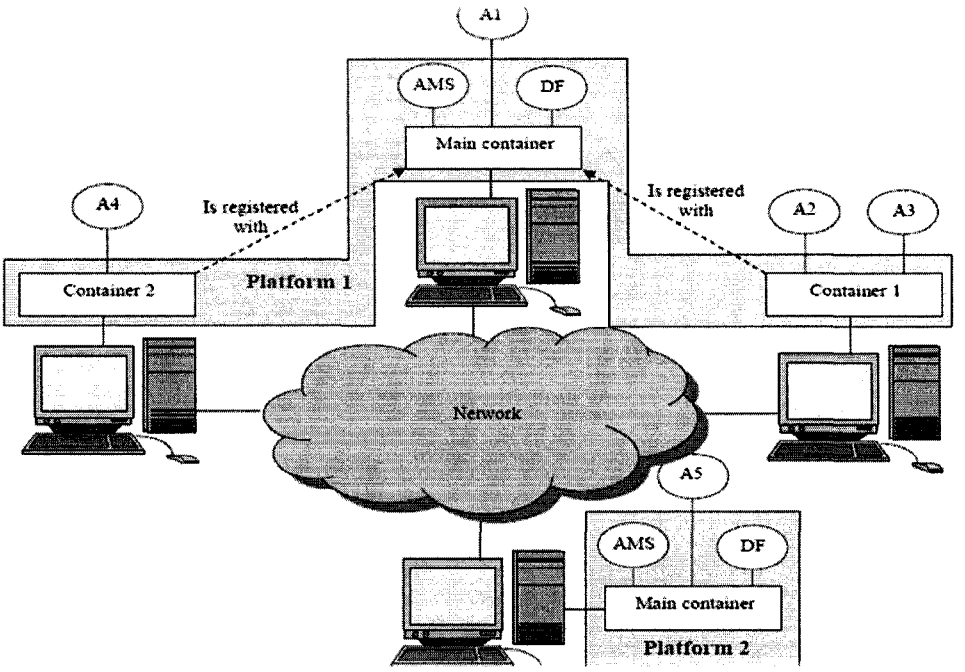

Figure 2.5 Architecture of Agent Platform

We can simply define a class extending the *jade.core.Agent* class and implement the *setup()* method to create a JADE agent. Each agent is identified by an "*agent identifier*" for example *<nickname>@<platform-name>*, which complies with FIPA agent identifier specification. A task that an agent can carry out is implemented by a behaviors class in JADE. An agent can execute several behaviors concurrently. Figure 2.6 depicts the execution path of the agent thread [38].

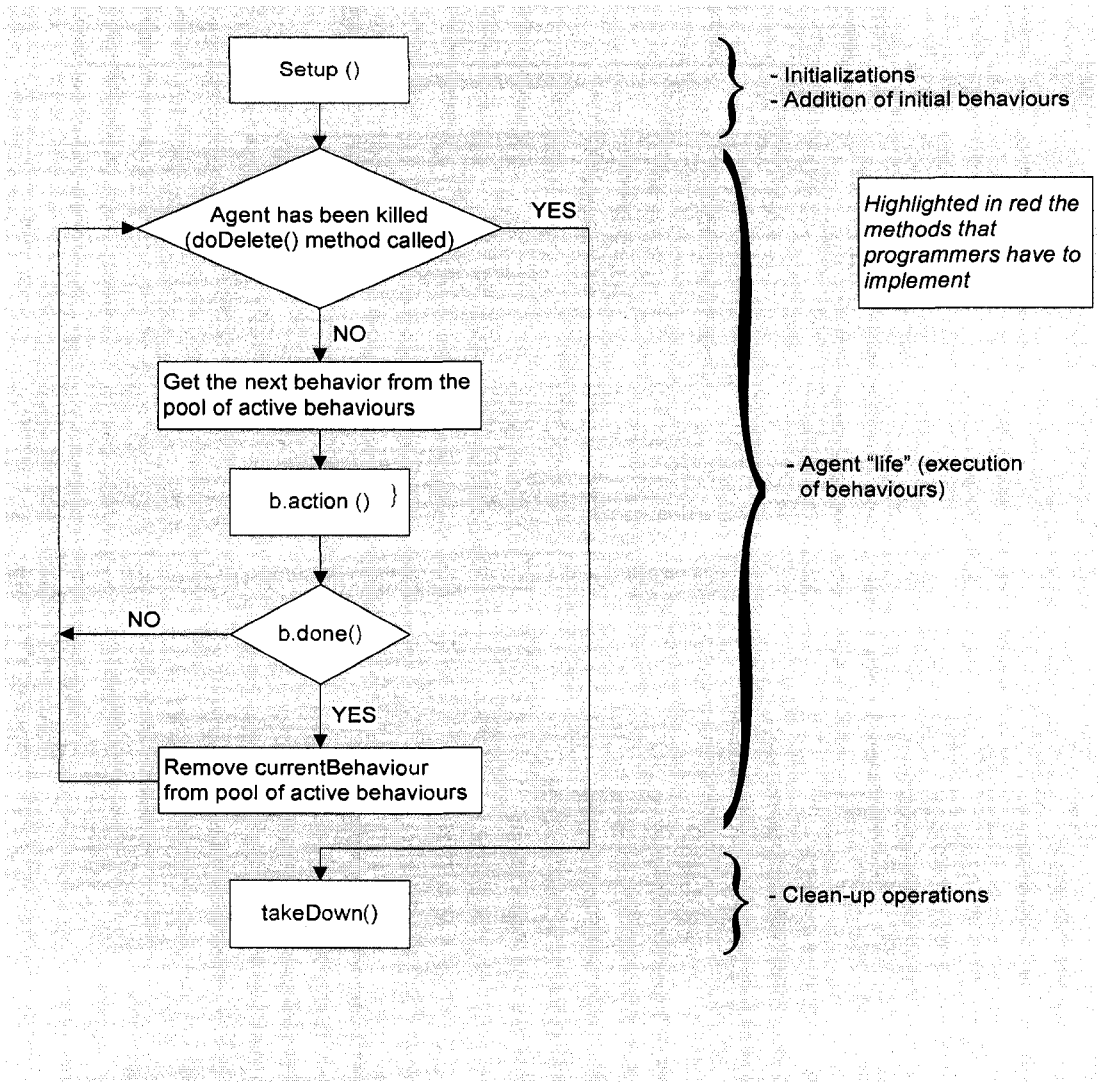

Figure 2.6 Agent Thread path of execution

In JADE, the structure of a message complies with the FIPA-ACL standard as well. It includes fields (or parameters) so that it can support complex interactions and multiple parallel conversations.

The whole JADE source code is distributed under an open source policy. The Web site http://jade.tilab.com/ provides many online documents.

2.4.2 Jadex BDI Agent System

Jadex is based on the JADE agent platform. Like JADE, Jadex is a java based, FIPA compliant agent environment. However, it allows developing goal oriented agent following the BDI model. The abstract Jadex architecture [38] is presented in figure 2.7. An agent can receive or send messages. The received message or goal events can trigger the agent internal reaction and deliberation mechanism, which dispatches the events to plans selected from the plan base. Running plans may access and modify the belief base, exchange message with other agents, create new goals, and cause internal events again.

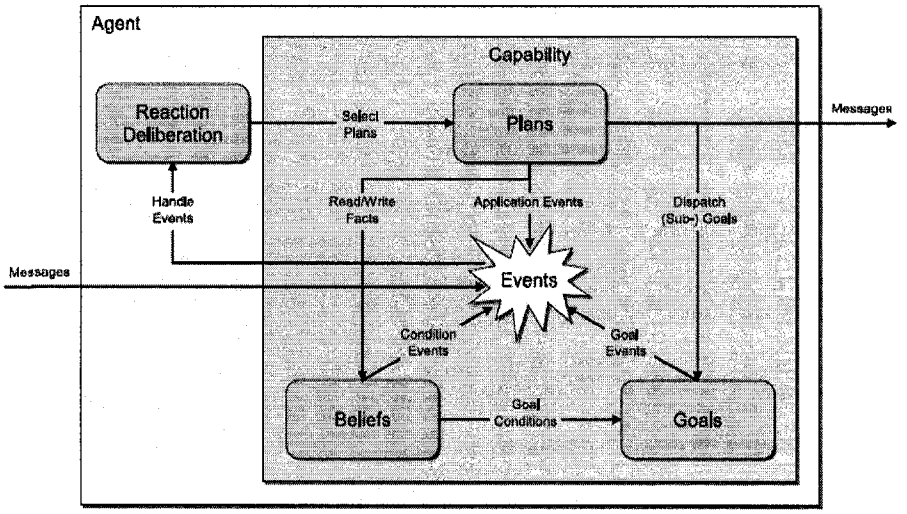

Figure 2.7 Jadex abstract architecture

The Beliefbase

The beliefbase, a set of an agent's belief, stores believed facts that make up the agent's knowledge. Unlike other BDI Systems, which beliefs are represented in some kind of first-order predicate logic (e.g. Jason) or using relational models (e.g. JACK), the belief in Jadex is very simple, just as storage of knowledge, like database of a agent. Belief currently does not support any inference mechanism. On top of this simple belief representation, Jadex adds several advanced features. Jadex uses an OQL (Object Query Language)-like query language that is

adopted from the object-relational database world to search the conditions that trigger plans or goals when some beliefs change [37]. Belief also can be stored as expressions and evaluated dynamically on demand.

The Goals Structure

In Jadex, goals are a central concept and not just a special kind of event like in pure BDI models. For any goal it has, an agent will engage into certain actions, until the goal has been reached, unreachable, or not desired any more. A goal lifecycle (Figure 2.8) consists of the goal states *option, active, and suspended* [38]. It will distinguish between just adopted and actively pursued goals. When a goal is adopted, it becomes an option that is added to the agent's desire structure. Application specific goal deliberation mechanisms are responsible for managing the state transitions of all adopted goals.

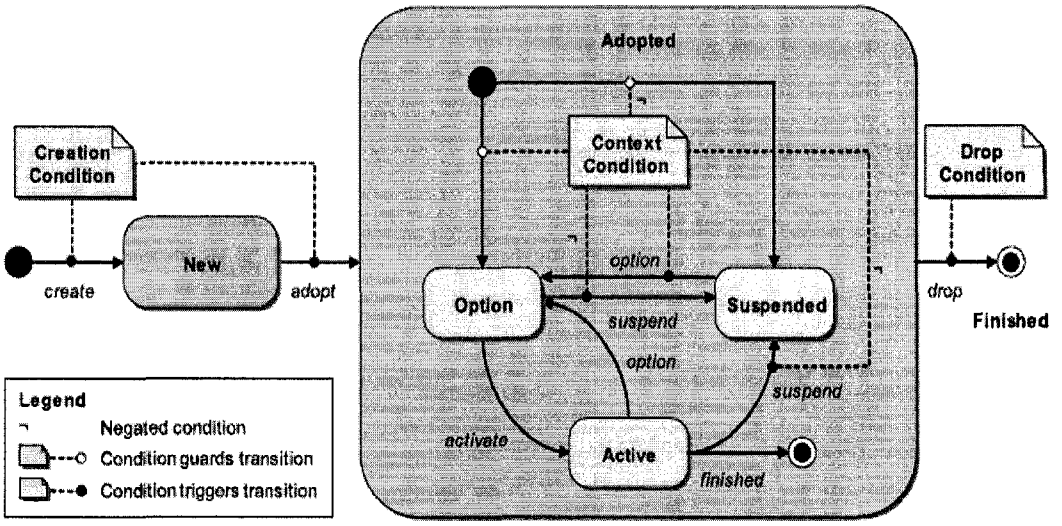

Figure 2.8 Goal lifecycle

Four types of goals, which extend the general lifecycle and exhibit different behavior with regard to their processing, can be distinguished: *Achieve*, *maintain*, *perform*, and *query*.

- An *achieve goal* just defines a desired target state, without specifying how to reach it.
- A *maintain goal* specifies a state that should be kept (maintained) once it is achieved.

- A *perform goal* states that something should be done but may not necessarily lead to any specific result.
- A *query goal* represents a need for information.

Plan Specification

Plans are used to specify the actions an agent may perform to reach its goals. Jadex uses the plan-library approach to represent the plans of an agent. Therefore, plans, usually be written in Java and predefined by the developer, compose the library of actions the agent can perform, Plans provide all the flexibilities of the Java programming language. Plans are instantiated to handle events and to achieve goals.

In Jadex, plans consist of two parts: A Plan head and a corresponding plan body. The plan head is declared the ADF whereas the plan body is realized in a concrete Java class. Hence, the plan head defines the circumstances under which the plan body is instantiated and executed. Depending on the current circumstance, plans are selected in response to occurring events or goals. The selection of plans is done automatically by the system and represents on main aspect of a BDI infrastructure.

Agent Definition

The complete definition of an agent is captured in an XML file, which is called *agent definition file* (ADF). The ADF contains the beliefs, goals, events, plans, and other agent elements and can be seen as a type specification for a class of instantiated agents. Plans are declared by specifying how to instantiate them from Java class. For example, Buyer agents are defined by the *Buyer.agent.xml* file, and file *PurchaseBookPlan.java*. construct plans implemented. Moreover, the initial state of an agent is determined in a configuration tag, which defines the initial beliefs, initial goals, and initial plans. In Jadex, The ADF is loaded first in order to start an agent, and the agent is initialized with beliefs, goals, and plans defined by configuration tag.

Execution Model

Figure 2.9 depicts Jadex Execution model. Before incoming messages in the message queue can be forwarded to the system, it has to be assigned to a capability, which is able to handle the message. If the message belongs to an ongoing conversation, an event for the incoming message is created in the capability executing the conversation. Otherwise, a suitable capability has to be found. Then the created event is subsequently added to the agent's global event list.

The dispatcher is responsible for selecting applicable plans for the events from the event lists. Jadex provides flexible settings to influence this event processing individually for event types and instances. As a default, messages are posted to only one single plan, while for goals, many plans are executed sequentially until the goal is reached or failed, when no more plans are applicable. After plans have been selected, they are placed in the ready list, waiting for execution. The execution of plans is performed by a scheduler, which selects the plans from the ready list.

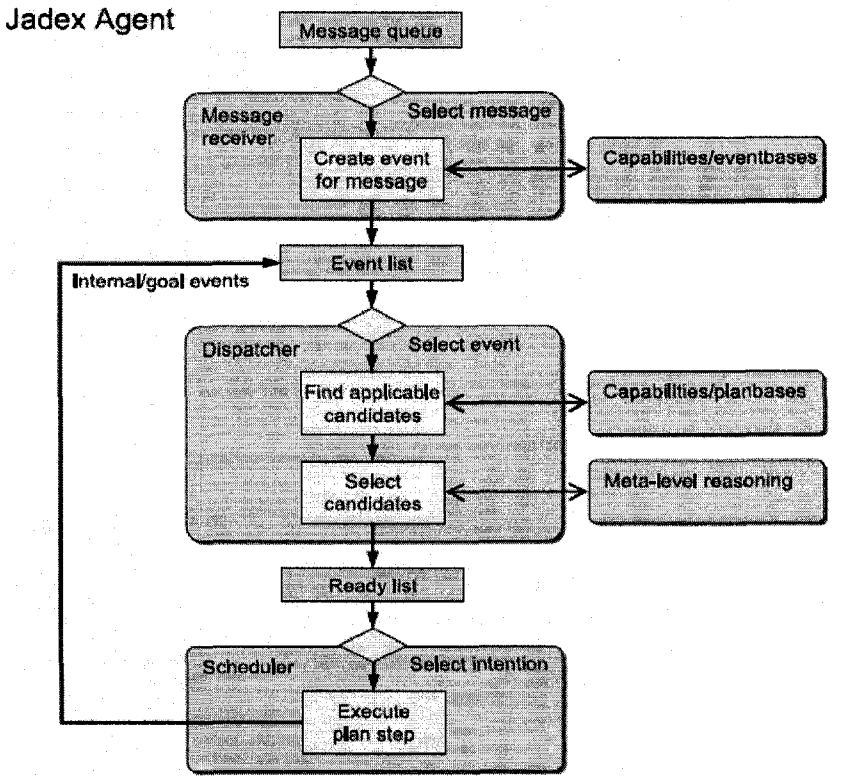

Figure 2.9 Jadex Execution model

2.5 Conclusion

In this chapter, we have provided a brief review of multi-agent systems and agent communication technology. There are three agent architectures: deliberative agent architecture, reactive agent architecture, and hybrid agent architecture. We mainly focused on the BDI architecture, one of the deliberative agent architectures. We reviewed KQML and FIPA-ACL for agent communication language. Furthermore, we introduced dialogue games to solve the problem that KQML and FIPA-ACL are not flexible to suit autonomous agents' negotiation.

We have also introduced Java-based multi-agent programming platforms: JADE and Jadex. Both JADE, a software framework to support the development of agent applications, and Jadex, a software framework for the reaction of goal-oriented agents following the belief-desire-intention (BDI) model, are FIPA compliant agent environment and open source projects around which a community of users and contributors has grown up. JADE provides the platform architecture and the core services and message transport mechanisms as required by the FIPA specifications. Hence, Jadex is based on the JADE Agent Framework to achieve FIPA-compliancy. Jadex allows developers to define more abstract goals for the agents, thereby providing a certain degree of flexibility on how to achieve the goals. This is the reason why we use Jadex as our implementation language. In the next chapters, we will discuss how we use Jadex to simulate our communities of Web services.

Chapter 3. Overview of Web Service Technology

This chapter presents an overview of Web service technology from two perspectives: standards for Web services and Web services interactions. Section 3.1 presents the main standards, namely UDDI, WSDL, and SOAP, and how these standards interoperate with each other. Section 3.2 focuses on Web services interactions and composition technology. It covers BPEL, Web components, semantic Web, and the emerging communities of Web services.

3.1 Basic Standards for Web Services

3.1.1 Generalities

Web services are self-contained and self-describing application components that communicate through open protocols with other applications for the purpose of using and exchanging data [44]. Web services enable users to access business functionalities, and they allow the integration of heterogeneous enterprise applications. Web services, normally with an XML interface, are registered and can be placed via a Web service registry. The general framework for Web services is shown in Figure 3.1. It contains three types of participants [41]: services providers, services registry, and services consumers.

- **Service provider**: A service provider is the party that creates Web services and advertises them to potential users by registering the Web servicers with service registry.
- **Service registry**: This part maintains a registry of advertised (published) services and might introduce service providers to service consumers.
- **Service consumer**: A service consumer searches the registries of service registry for suitable service providers, and then contacts a service provider to use its services.

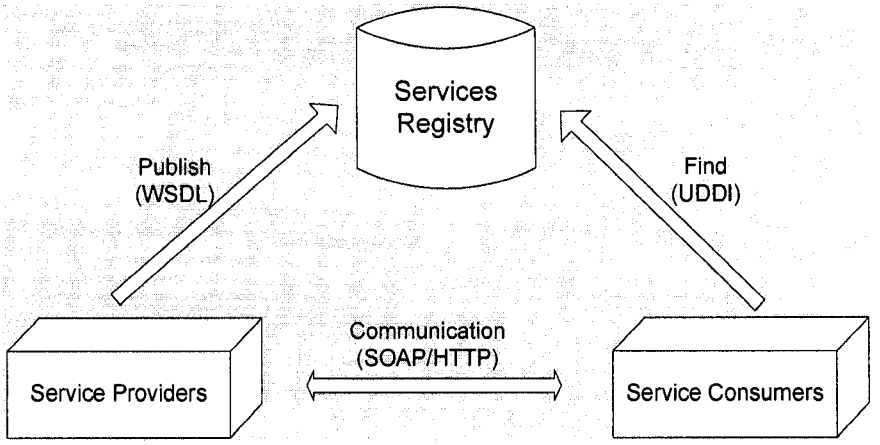

Figure 3.1 Framework fro Web services

The framework for Web services is founded on principles and standards for connection, messaging protocol, description, and discover. The eXtensible Markup Language (XML) provides a common language between the service providers and service consumers. The basic standards for Web services are defined by Web languages such as UDDI (Universal Description, Discovery, and Integration) [45], WSDL (Web Services Description Language) [17], SOAP (Simple Object Access Protocol) [40][48]. In this section, we will introduce the key elements of the three basic standards, SOAP, WSDL, and UDDI, for Web services.

3.1.2 Simple Object Access Protocol (SOAP)

SOAP is a message-based protocol for accessing services on the Web, normally using HTTP/HTTPS as the transport protocol and XML for data encoding. However, other transport protocols, such as FTP, SMTP, TCP/IP sockets, may also be used [44]. It was originally provided for networked computers with Remote-Procedure Call (RPC), in which one network node (the client) sends a request message to another node (the server) and the server immediately sends a response message to the client. Therefore, two types of messages, Request and Response, are defined by SOAP. Two parts, a header and the XML payload, comprise a SOAP message. The header content is based on different transport layer; nonetheless, the XML payload remains the same. The XML part of the SOAP request consists of three main portions:

- **Envelop**: this defines the various namespaces that are used by the rest of the SOAP message.
- **Header**: this is an optional element for carrying auxiliary information for authentication, transactions and payment.
- **Body**: this is the main payload of the message. It normally contains the method name, arguments and Web Service target address.

SOAP makes the security and communication behind proxies and firewalls easier than previous remote-procedure call because it is a character-based, rather than a binary protocol [41]. However, like every coin has two sides, SOAP is inefficient for many applications. Unfortunately, SOAP is not a state protocol from conceptual, nor describes bidirectional or multiparty interaction. Therefore, SOAP is effective for simple interoperability between single clients and servers, but for more complex interoperability among heterogeneous systems, a message-queuing component should be used by each participant to provide transaction and security support.

3.1.3 Web Services Description Language (WSDL)

Web Services Description Language (WSDL) is an XML-based language for specifying Web services by defining messages that provide an abstract definition of the data being transmitted and operations that a Web service provides to transmit the messages [43]. Developed by Microsoft and IBM, WSDL describes the protocols and formats used by the service. A WSDL document contains seven elements in the definition of network services. They are *Type, Message, Operation, Prot Type, Binding, Port, and Service.* Four *operation* types, which characterize the behaviour of an endpoint, are defined from the perspective of the ultimate implementation of the Web service.

- **One-way**: the endpoint receives a message.
- **Notification**: the endpoint sends a message.

- **Request-response**: the endpoint receives a message and sends a correlated message
- **Solicit-response**: the endpoint sends a message and receives a correlated message.

A WSDL specification is split into two main components: the interface and the implementation in order to promote reusability and having multiple implementations [41]. The WSDL interface describes a service by fleshing out the definition of WSDL elements. It is the abstract module and may import other interfaces. The WSDL service implementation focuses on the specifics of binding a service.

3.1.4 Universal Description, Discovery, and Integration (UDDI)

UDDI defines an online registry to publish information about businesses and services. It provides white-page and yellow-page services. Service consumers can use this registry information to find companies in a given industry with a given type of service and to locate information they need. The UDDI specifications comprise an XML schema for SOAP messages and a description of the UDDI APIs specification.

The information fields in UDDI white pages are business name, text description, contact information, and identifiers that a business may be known by. The yellow pages comprise business categories organized as three major classes [41]:

- Industry: North American Industry Classification System (NAICS), a six-digit code maintained by the US government for classifying companies.
- Products and services: Ecma International and United Nations Standard Products and Services Code
- Geographical location: ISO 3166 for country and region codes.

The UDDI specifies two APIs: *Inquiry API* and *Publish API* that contain message for interacting with UDDI registries. The *Inquiry API* retrieves information, for example to locate businesses, services, and bindings, from a registry. The *Publish API* stores information for creating and deleting UDDI data in the registry. The UDDI APIs are based on SOAP.

In fact, a UDDI business registry is a Web service based on XML and SOAP. IBM and Microsoft are UDDI Registries providers. A business registry provides operations to create, modify, delete, and query of data structures. All these operations can be performed either via a Web site or by using tools that make use of the UDDI API specification.

3.1.5 Interoperation of SOAP, WSDL and UDDI

WSDL descriptions can be housed in a UDDI directory, and the combination of WSDL and UDDI is expected to promote the use of Web services worldwide. Tsalgatidou [44] has abstracted SOAP, WSDL, and UDDI integration and interoperability into layers (Figure 3.2). Based on common internet protocols HTTP, TCP/IP, etc., XML provides a cross-platform approach to data encoding and formatting. SOAP and WSDL, which are built on XML, define a simple way to package information for exchanges across system boundaries and specifies properties of a Web services. The UDDI provides a stage that allows two different services to share each other's information and to describe their own services. The top layer, Universal Service Interop Protocols, will be defined in the future more advance discovery features, such as the ability to locate parties that can provide a specific product or service at a given price or within a specific geographic boundary in a given timeframe.

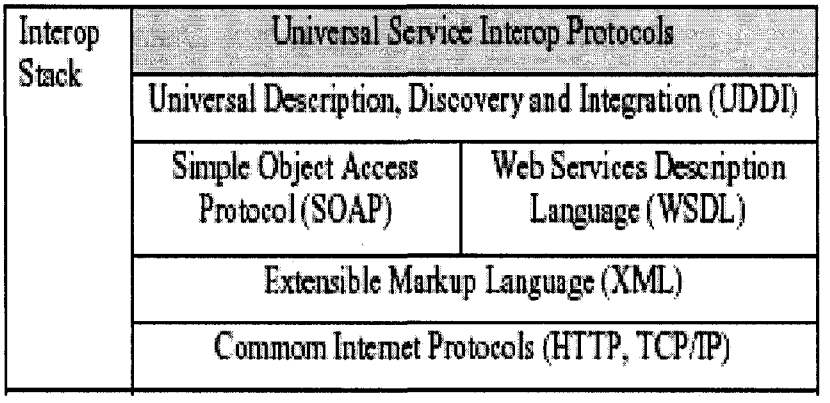

Figure 3.2 Layered view of SOAP, WSDL, and UDDI standards (from [44])

3.2 Web Services Interactions

3.2.1 Generalities

The Web Services are internet-enabled applications capable not only of performing business activities on their own, but also possessing the ability to engage other Web services in order to complete higher-order business transactions. However the Web service standards that we introduce in the previous section do not deal with the dynamic composition of existing services notwithstanding [42]. Emerging Web service standards are clear evidence that the Web service community is inexorably moving toward representing compositions where flow of the process and the bindings between services are known a priority.

A conversational model for Web services provides a more loosely coupled, peer-to-peer interaction model. By interacting, composite Web services are able to manipulate complex transactions that can be satisfied, not by one single available resource, but by a combination of efforts through business to business collaboration among these applications. It accelerates rapid application development, service reuse, and complex service consummations. From user's perspective, service composition, which compositie the different web services for users, offers seamless access to a variety of complex services. Nonetheless, service composition is not standardized and there are no set principles to ensure that quality of service (QoS) requirement are met.

There are several existing approaches for Web services interaction and composition, such as Business Process Execution Language (BPEL[3]), Web components, Petri nets, algebraic process composition, finite-state machines, semantic Web (OWL-S), and the emerging communities of Web services. These approaches are discussed below.

[3] www.ibm/com/developerworks/library/ws-bpel

3.2.2 BPEL (Business Process Execution Language)

BPEL is an XML-based language for the formal specification of business processes [20] and business interaction protocols. BPEL extends the Web services interaction model and enables it to support business transactions. It is developed by BEA, IBM, Microsoft, SAP, and Siebel.

BPEL composition collaborates with different Web services to achieve the task that the user pursues. A process, which names the composition result in BPEL, has two forms: an executable process, which specifies the execution order, and an abstract process, which defines negotiation protocols, specifying the exchanged messages among different participants in the orchestrated process. A process is defined by a BPEL source file, which describes activities, a process interface, which describes ports of a composed service, and an optional deployment descriptor, which contains the partner services' physical locations [31].

Unfortunately, in BPEL multiple service composition is somewhat tedious when XML files start to grow. This does not follow the aim of abstract process and it also complicates the protocol description, revealing unnecessarily implementation details and making difficult the task of analyzing the correctness. Furthermore, BPEL has no standard graphical notation [31] although there are proposals to use UML-like notation for description, and it provides a static composition Web service.

3.2.3 Web Components

Web components are a packaging unit for developing Web-based distributed applications in order to combine existing Web services. After a Web component class is defined, Web components can exchange services as components for supporting basic software development principles such as reuse, specialization, and extension [53].

Composite logic, which is the way a composite service constructed by its constituent services, is comprised of composition type and message dependency. The composition type can

be in the form that determines in what order the messages are executed or in the form of alternative execution, which indicates whether a component can invoke alternative services until one succeeds. Message dependencies can be in the form of synthesis that combines output messages of constituent services, decomposition that binds the input messages of composed services into the input messages, and message mapping that allows custom mapping between constituent services' input and output. Web components can be specified in two isomorphic forms: a class definition and an XML specified and described in Service Composition Language that consists of the composite services interface and the composition logic.

Web components offer both compatibility and conformance checking. This approach achieves good scalability with class definitions, but requires additional time for mapping and synchronization between class definitions and XML.

3.2.4 Semantic Web

Semantic Web, sometimes called "Web 3.0", uses XML tags that conform to Resource Description Framework (RDF [4]) and Web Ontology Language formats (OWL [5]). These technologies are combined in terms of providing descriptions that supplement or replace the content of Web documents. Thus, with semantic Web, Web content can be expressed not only in natural language, but also in a format that can be read and used by software agents, thus permitting service consumers to find, share and integrate information more easily. In other words, Web service users and software agents are able to discover, compose, and invoke content using complex services easily.

The semantic Web consists of several standards and tools. The components are shown in W3C Semantic Web Layer (Figure 3.3). The OWL describes the functions and relationships of each of these components of the semantic Web.

---

[4] http://www.w3.org/RDF
[5] http://www.w3.org/TR/owl-features/)

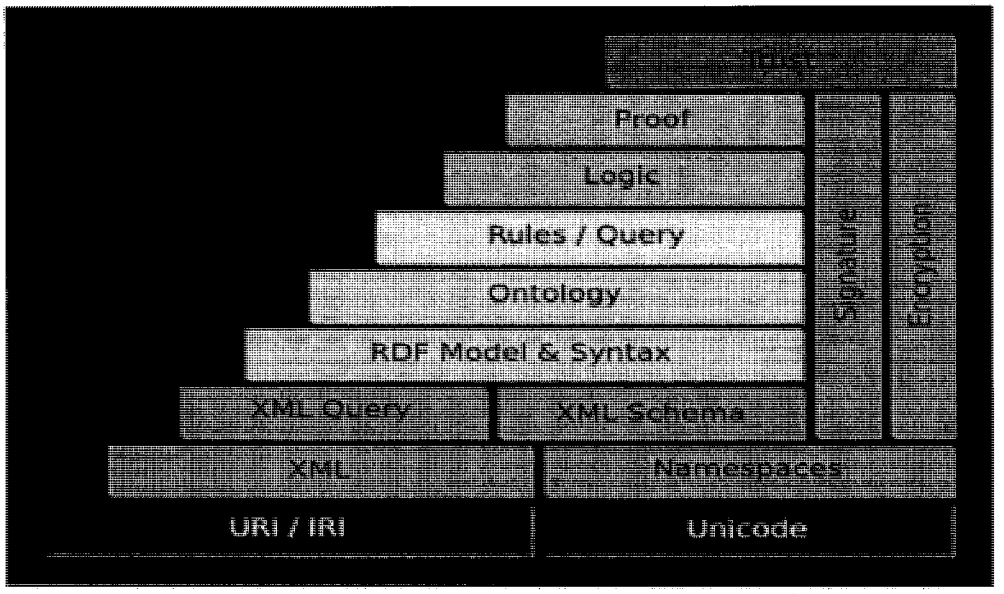

Figure 3.3 W3C semantic-Web-layers
(From http://www.w3.org/2006/Talks/1023-sb-W3CTechSemWeb/Overview.html#(19)

An RDF description, written in multiple notations (e.g. XML, Notation3), is a set of triples that each triple is akin to the subject, verb, and object of a sentence and its element is represented by a Universal Resource Identifier (URI) [43]. RDF and RDF Schema express classes, properties, ranges and documentation for resources and the ontology to represent further relationship and /or properties like equivalence, lists, and data types.

The OWL defines and instantiates service ontology that enables automatic services discovery, invocation, composition, and execution monitoring. OWL models a service using a *service profile,* which describes the service requirements from the user and vice, versa, a *service model,* which specifies the service, and a *service grounding,* which gives information on how to use the service.

The process model, including processes and their dependencies and interactions, is a service model that describes a service in terms of inputs, outputs, pre-conditions, and post-conditions. The Semantic Web provides a process level description of the service so that the

evolution of the domain can be logically inferred. It relies on ontology to formalize domain concepts which are shared among services.

3.2.5 Communities of Web Services

It is clear that the Web service composition is moving toward "agent-like" model, largely independent of the traditional software agent community. Interoperation between agents and Web services is interesting because Web services need for autonomy, heterogeneity, and dynamism from agents. Unlike conventional services, agent-based services know about themselves, their users, and their competitors. Furthermore, they use and reconcile ontology, they are proactive and communicative, and they can be cooperative.

Based on both Web services and multi-agent technologies, a new approach for interacting Web services was proposed by J. Bentahar, et al [8]. In this approach, Web services are organized in communities using argumentative agents. The idea is to gather Web services with similar functionalities into special structures in order to make them cooperate and to promote their participation in composite scenarios. A community is organized dynamically by agents with a specific protocol for community [6]. In such a protocol, Web services can *accept, challenge, attack, and refuse* proposals related to their joining different communities and their participation in composite business scenarios. The details of this protocol will be discussed in the next chapter.

The architecture of such structure is shown in Figure 3.4. The components in this architecture include service providers, service consumers, and service registries. The figure shows two communities as example: Map Services and Traffic Information communities. Service providers publish and register their services in Services Registry so that service consumers or users can search for them. There are two kinds of argumentative agents in a community: Master Web Service Agent (Master-WS) and Slave Web Service agent (Slave-WS). A Master-WS always leads a community. It can be implemented as a Web Service for compatibility purposes with the rest of Web services that populate the community as well. Slave-WSs have in common

the functionality of the community to which they belong. Slave-WSs compete to participate in composition scenarios in the same community because they all achieve the same or similar functionality but in a different settings.

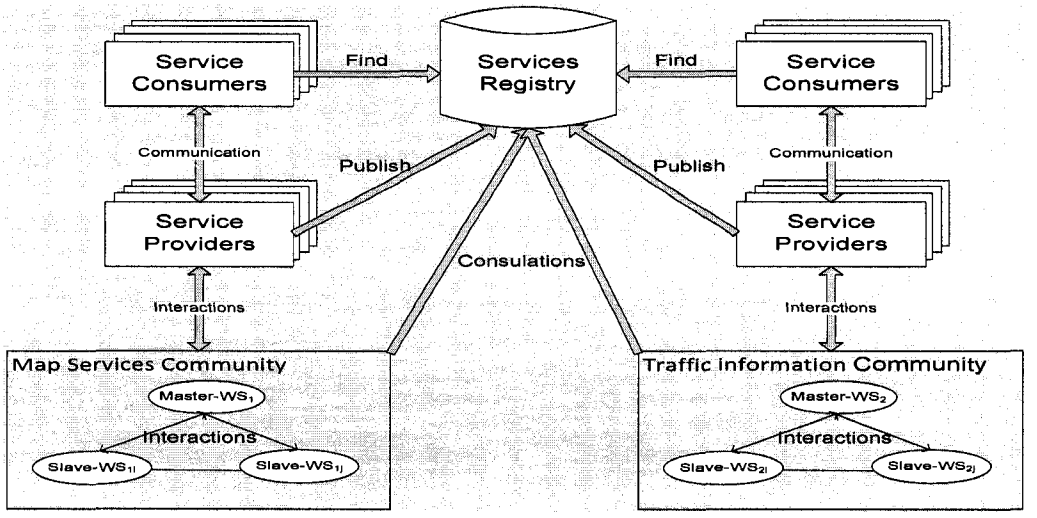

Figure 3.4 Architecture of an environment of several Web Service communities

Several operations should be specified to establish and maintain a community of Web services. First, the functionality of the community should be defined by binding to a specific ontology [23]. This will decide the terminologies to be used in order to describe the functionality of the community. Then, the master Web service that leads the community and takes over the multiple responsibilities to manipulate and to maintain the community should be deployed. The responsibilities of the master Web service include selecting and inviting Web services to join the community and negotiating the rewards with those Web services. To select a Web service, a master Web service needs to check the credentials of a Web service, which could be related to QoS, protection mechanisms, interaction protocols, etc. For maintaining a high quality community, a master Web service will monitor the slave Web services' activities in the community. If the number of Web services in the community is less than a certain threshold and

the number of participation requests in composite Web services that arrive from users over a certain period of time is less than another threshold, the community could be dismantled.

3.3 Conclusion

The framework for Web services comprises three parts: service providers, service registry, and service consumers. We first introduced basic standards, like SOAP, WSDL, and UDDI for Web services. SOAP provides the common communication protocol to connect service consumers and services providers. WSDL provides description of Web services for publishing service providers to service registry. Service consumers search the services they need through UDDI. Then we discussed Web service interactions and composition technology, such as BPEL, semantic Web, Web components, and communities of Web services.

Although the work reviewed in this chapter about Web service technologies covers basic standards and Web services interactions, there is still a plenty of room for further research, especially for communities of Web service. For example, how we can design efficient communities of Web services, considering both structural and QoS properties of required Web services? Also, how we can specify, verify, and implement the Web service communities? These questions will be considered in Chapters 4 and 5 which are about our contributions.

Chapter 4. Specifying and Implementing Communities of Web Services

We have presented the architecture of Web services communities in Chapter 3. In this chapter, we specify communities of Web services using argumentative agents and implement the specified model with Jadex. Argumentative agents in a community, usually led by a master Web service, are able to persuade and negotiate with each other. By communicating, Web services organize themselves in a better way so they can achieve the goals they set in an efficient way.

This chapter is structured as follows. We first discuss the underlying management operations, and show through concrete scenarios the advantages of using argumentative Web services for these management operations in Section 4.1. Then, in Section 4.2, we present the argumentative agent-based framework for these communities. Sections 4.3 and 4.4 present a persuasive negotiation protocol that agents use to manage their communities. In Section 4.5, we discuss the formal properties of the protocol along with its computational complexity. We present our implementations method and some experimental results in Sections 4.6 and 4.7. Finally, we conclude this chapter in Section 4.8.

4.1 Management Operations for Communities of Web Services

The architecture of communities of Web services have been discussed in Chapter 3. Based on the architecture presented in Figure 3.4, we analyze the management operations for communities of Web services in this section. These operations revolve around developing/dismantling a new/existing community, attracting new Web services to be enrolled in an existing community, and retaining existing Web services in a community.

4.1.1 Community of Web Services Development

A community is initially deployed to gather Web services with a similar functionality. This deployment occurs in two steps. First it defines the functionality (e.g., FlightBooking) of the community by binding to a specific ontology [22]. This binding is important since providers of Web services use different terminologies to describe the functionality of their respective Web services. For example, *FlightBooking, FlightReservation, and AirTicketBooking* are all about the same functionality. The description of a Web service's functionality needs to be mapped onto the description of the functionality of the community using a specific ontology (i.e., ontology consists of concepts, axioms, relations, and instances). We assume that our Web services are mapped to a special ontology. However, this issue is out of the thesis scope.

The second step in establishing a community is to deploy the master Web service of the community to take over the multiple responsibilities. Some of these responsibilities include inviting Web services to sign up in its community and checking the credentials of Web services before they are admitted in its community. Credentials could be related to QoS (latency, execution time, privacy and security mechanisms, reliability, integrity, etc.), interaction protocols, interoperability, etc. Credential checking is critical to the reputation of a community as this boosts the security level in a community and enhances the trustworthiness level of a master Web service towards its slave Web services.

Dismantling a community of Web services happens upon request from the master Web service. If this latter notices that the number of Web services in the community is less than a certain threshold and the number of participation requests in composite Web services that arrive from users over a certain period of time is less than another threshold, then the community will be dismantled. Both thresholds are set by the designer. A slave Web service that is ejected from a community is invited to join other communities subject to assessing the similarities between functionalities.

4.1.2 Web Services Attraction and Retention

Attracting new Web services to a community and retaining the existing Web services in a community are responsibilities of the master Web service as well. A community of Web services could disappear if the number of residing Web services drops below a certain threshold. Attracting Web services drives the master Web service to regularly consult the different UDDI registries looking for new Web services. These latter could have recently been posted on an UDDI registry or have seen their description changed. Changes in a Web service's description raise challenges since a Web service may no longer be appropriate for a community. As a result, this Web service is invited to leave the community. When a candidate Web service is identified in an UDDI registry according to its functionality, the master Web service interacts with this candidate. The purpose is to persuade the candidate Web service to register with its community. An argument that is used during this interaction is the high rate of participation of the existing Web services in composition scenarios, which is a good indicator of the visibility of a community to the external environment. Other arguments include short response-time in handling users' requests, and efficiency of the security mechanisms against malicious Web services.

Retaining Web services in a community for a long period of time is a good indicator of the following elements:

- Although the Web services in a community are in competition, they expose a cooperative attitude. For instance, Web services are not subject to attacks from peers in the community. This backs the security argument that the master Web service uses to attract new Web services.
- A Web service is to a certain extent satisfied with its participation rate in composite Web services. This satisfaction rate is set by the provider of the Web service. In addition, this is inline with the participation-rate argument that the master Web service uses to attract new Web services.

- Web services are aware of peers in the community that could replace them in case of failure with less impact on the ongoing composite Web services in which they participate.

Web services attraction and retention shed the light on a scenario. It is about Web services that are asked to leave a community. A master Web service could issue such a request upon assessment of the following criteria:

- The Web service has a new functionality, which does not perfectly match the functionality of the community.
- The Web service is unreliable. In different occasions, the Web service failed to participate in composite Web services due to recurrent operation problems.
- The credentials of the Web service were "beefed up" to enhance its participation opportunities in compositions. Ouzzani and Bouguettaya [35] report that a Web service may not always fulfill its advertised QoS parameters due to various fluctuations related, for example to the network status or resource availability. Therefore, some differences between advertised and delivered QoS values occur. However, large differences indicate that the Web service is suffering a performance degradation and might not be able to sustain its advertised QoS.

4.2 Specification for Argumentative Communities of Web Services

4.2.1 Formal Foundation

The characteristics of argumentation-based agents discussed in Chapter 2 make agents suitable for modeling dynamic and proactive Web services. The primary value-added is to let Web services interact and argue with each other before joining and settling down in a community. They will be able to reason about and compare different joining offers in order to maximize their benefits. In addition, they can maximize their performance within the community by negotiating

with peers the participation conditions in composite scenarios. They can share participation benefits and collaborate by sharing resources and replacing each other if some problems arise at run-time [2].

Our argumentative agents act as representatives' to Web services, reason on their behalf, and identify situations that maximize their profits (e.g., participation-rate increase) and minimize their expenses (e.g., resource consumption decrease). Metadata describing Web services in terms of contents and features are represented within the state of the agents. Web services within communities are connected through a communication network so that they can interact and share resources to reach some joint goals.

A persuasive negotiation protocol allows argumentative Web services to negotiate the contract of joining a community and their participation in composite scenarios. To reason about Web services and communities, argumentative agents are equipped with knowledge, beliefs, and argumentation capabilities. The agent of a Web service *Agws* knows all details on its Web service in terms of functionality, QoS, (mean response time, execution time, transaction time, throughput, reliability, integrity, etc.), and any other relevant details. The knowledge base of *Agws* is denoted by *KB*(*Agws*). An argumentative Web service can also have beliefs towards other Web services whether in the same community or in other communities. Descriptions of these Web services, their functionalities, QoS, and trust are examples of these beliefs. As explained in the previous section, the agent's argumentation system is built upon the agent's beliefs and knowledge.

In [36], Parsons et al. prove that argumentation reasoning procedures based on languages expressed in the first order logic or even propositional logic are computationally intractable. For communities of Web services, a restricted language such as *Horn logic* is enough to represent agents' beliefs and to develop their reasoning capacities. Horn logic is expressed in terms of *propositional Horn clauses*. Such a clause is a disjunction of literals with at most one positive

literal $\neg p_1 \vee \neg p_2 \vee ... \vee \neg p_n \vee c$ (also written as implication $p_1 \wedge p_2 \wedge ... \wedge p_n \rightarrow c$). *A propositional Horn formula* is a conjunction of propositional Horn clauses. These clauses could be restricted to be *definite* where each clause has exactly one positive literal. *A propositional definite Horn formula* is a conjunction of propositional definite Horn clauses. This restriction is of a particular interest in modeling argumentative reasoning, since formulas of type $p_1 \wedge p_2 \wedge ... \wedge p_n \rightarrow c$ are adequate to describe interrelationships between premises and conclusions. This could be used to support positive literals. For example, the master Web service can have an argument supporting the conclusion that the community it is leading has a good reputation, which can be represented as follows:

$$NWS = High \wedge NUR = High \wedge NSR = High \wedge NLW = Low \rightarrow reputation = Good$$

where *NWS* stands for Number of current Web Services in the community, *NUR* for Number of Users' Requests the community receives by time unit, *NSC* for Number of Successful Compositions by time unit to which the community's Web services participate, and *NLW* for Number of Leaving Web services. Also, a slave Web service can have an argument supporting the fact that it deserves a better offer, which can be represented as follows:

$$R = V_1 \wedge R\_C_x = V_2 \wedge V_2 \geq V_1 \wedge QoS = Good \rightarrow DR \geq V1$$

where R stands for the Reward offered by the community the Web service is negotiating with, R_C_x for the Reward another Community (called Cx) already offered to this Web service, and *DR* for Deserved Reward. In other words, when negotiating with a given community, if an argumentative Web service has a good QoS and already received a competitive offer from another community, then it has an argument to ask for a better offer. In addition, for Web services, there is no need to suppose that the knowledge bases are inconsistent. The reason is that the size of these knowledge bases is generally small enough, so that checking the consistency when a new belief is added becomes doable in a linear time.

4.2.2 General Specification

To be able to persuade an argumentative Web service to join a community or to remain in a community, and to negotiate the participation in a given composite scenario along with the outcome of the contract-net protocol, the master and slave argumentative Web services use persuasive negotiation techniques based upon their argumentation abilities. Hereafter, we specify a Horn logic-based protocol to use for these persuasion and negotiation activities. This protocol is specified as a combination of a set of initiative/reactive *dialogue games*. Dialogue games can be thought of as interaction games in which each Web service plays a move in turn by performing utterances according to a pre-defined set of rules. Dialogue games have the advantage of being more flexible than classical protocols such as FIPA-ACL protocols. Indeed, a dialogue game can be specified as a combination of small conversation policies that Web services can combine by reasoning over them using a set of logical rules [5]. From a logical point of view, game moves are considered as communicative acts that argumentative Web services perform. Formally, we define a protocol for argumentative Web services as follows:

Definition 4.1 (Protocol). *A protocol Pr for argumentative Web services is a tuple <C,D> where C is a finite set of allowed communicative acts and D is a set of dialogue games.*

The allowed communicative acts in our persuasive negotiation protocol are: *Open*, *Accept*, *Refuse*, *Make-Offer*, *Challenge*, *Justify*, and *Attack*. *Open* is a special communicative act used to open the protocol. The type of a communicative act refers to its name, for example *Accept* is a communicative act of type *Accept*. We define a dialogue game in our protocol as follows:

Definition 4.2 (Dialogue Game). *Let* $CA_i(Agws_1, Agws_2, p)$ *be a communicative act of type i performed by an argumentative Web service* $Agws_1$ *and sent to another argumentative Web service* $Agws_2$ *about a content p, and* $CA_{ij}(Agws_2, Agws_1, p')$ *be the communicative act of type j that depends on the communicative act of type i. A dialogue game Dg is a conjunction of rules, where each rule identifies one possible communicative act that a Web service can use as a reply*

when receiving a communicative act from another Web service if a given condition C_{ij} is satisfied. This conjunction is specified as follows:

$$\bigwedge_{0<j<n} (CA_i(Agws_1, Agws_2, p) \wedge C_{i_j} \Rightarrow CA_{i_j}(Agws_2, Agws_1, p'))$$

where n is the number of allowed communicative acts that Agws2 can perform after receiving a communicative act from Agws1.

In this definition, content p could be a Horn formula or an argument expressed in Horn clauses. C_{i_j} is expressed in terms of the possibility of generating an argument from the argumentation system.

The formula $p \lhd Arg\_Sys(Agws)$ expressed in Horn language $\mathcal{L}$ denotes the fact that a Horn propositional formula p can be generated from the *Agws*'s argumentation system denoted by *Arg_Sys(Agws)*. The formula $\neg (p \lhd Arg\_Sys(Agws))$ indicates the fact that p cannot be generated from *Agws*'s argumentation system. For example, if the master Web service *AgMWS_Cx* of a community *Cx* has an argument for the fact that the reputation of its community is good, this will be represented by: *Reputation_Cx = Good* $\lhd$ *Arg_Sys(AgMWS_Cx)*. A Horn propositional formula p can be generated from an argumentative Web service's argumentation system, if this Web service can build an argument supporting p using its argumentation system. The following is an example of a dialogue game, in which a master Web service *AgMWS1_Cx* of a community *Cx* invites a Web service *Agws2* to join the community.

Example 1.

$$Open(Ag_{MWS1}\_Cx,\ Ag_{WS2}, p) \wedge (\neg\ (Reputation\_Cx = Poor \lhd Arg\_Sys(Ag_{WS2}))$$

$$\wedge\ (Joining\_Commitment = false \lhd Arg\_Sys(Ag_{WS2})))$$

$$\Rightarrow Accept(Ag_{WS2}, Ag_{MWS1}\_Cx, p)$$

where $p = Invitation\_for\_Joining\_Cx$

In this example, *AgWS2* accepts the invitation because it does not have any argument supporting the fact that the reputation of the community is "poor" and it has an argument that it is

not committed to join any other community (*Joining Commitment = false*). Accepting the invitation does not mean that $Agws_2$ commits to join the community, but only a negotiation of the joining contract can start.

4.3 Argumentative Dialogue Games for Communities of Web Services

In our persuasive negotiation protocol for argumentative Web services, we distinguish three types of dialogue games (Figure 4.1): *Entry game*, *Chaining games*, and *Termination game*. The *Entry game* enables conversation opening and setting up. The *Chaining games* make it possible to continue the conversation by combining several dialogue games. The persuasive negotiation protocol includes four chaining dialogue games: *Offer game*, *Challenge game*, *Attack game*, and *Justification game*. The conversation terminates when the exit conditions are satisfied (*Termination game*).

$$(Open(Agws_1, Agws_2, p) \wedge C_1 \Rightarrow Accept(Agws_1, Agws_2, p)$$

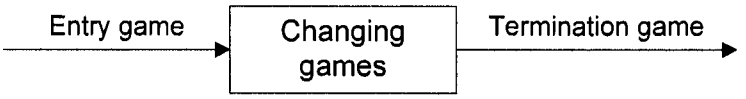

Figure 4.1 Types of dialogue games

4.3.1 $\mathcal{E}$ntry Game

The *$\mathcal{E}$ntry game* allows Web services to initiate conversations. For example, if a master Web service decides to invite a new Web service registered in a given UDDI to be a member of its community, this master will trigger an *$\mathcal{E}$ntry game* with invitation to join the community as subject. If the new Web service accepts, then the master can suggest rewards to the Web service if its final decision is to join the community. If the Web service refuses the invitation, the protocol terminates. A Web service can turn down invitations if it is not interested in a community (e.g., low participation-rate of existing Web services, or decided to join another

community). Within a same community, a Web service can invite other Web services to negotiate their participation in a composite Web service. When several agents provide the same "winning" bid following the master Web service's call for bids. The master asks one of the winnings to invite others for negotiation. The negotiation starts upon receiving and accepting this invitation. We specify the *Entry game* as follows:

$$(Open(Agws_1, AgWS_2, p) \wedge C_1 \Rightarrow Accept(Agws_2, AgWS_1, p))$$
$$\wedge (Open(Agws_1, AgWS_2, p) \wedge C_2 \Rightarrow Refuse(Agws_2, AgWS_1, p))$$

where:

$$C_1 = (p \lhd Arg\_Sys(AgWS_2)) \vee \neg (\neg p \lhd Arg\_Sys(AgWS_2))$$
$$C_2 = \neg p \lhd Arg\_Sys(AgWS_2)$$

Proposition p is expressed in the logical language $\mathcal{L}$ using a shared ontology. This proposition indicates an invitation to start a conversation. If the invited Web service has an argument in favor of p or does not have any argument against p, it accepts the invitation. Otherwise, it refuses. For example, if a new Web service is not interested in joining a community due to previous unsuccessful experiences in this community, a refusal is sent to the master Web service. If a Web service believes that the community's configuration is efficient (i.e., good reputation and high mean participation rate), and no commitment is made to join any other community, then it will accept the invitation.

4.3.2 Offer Game

Once the *Entry game* is accepted, the initiator argumentative Web service starts by making an offer. In the case of inviting a new Web service to join a community, the offer contains the initial rewards that the master offers to the new Web service and the advantages of being a member of this community. In the case of persuading an existing Web service to remain in the community, the offer could include increases in the rewards. In the case of negotiating a participation in a composite Web service, the offer contains the rewards that the initiator will give

to the other Web service after the composition, for example, a part of the rewards this initiator will obtain after its participation. Let p and q be two Horn formulas representing offers (contents of an *Offer* communicative act). The notation $p \ ' \ q$ indicates the fact that these two offers are for the same object. We specify the *Offer game* as follows:

$$(Make\text{-}Offer(Agws_1, Agws_2, p) \wedge C_1 \Rightarrow Accept(Agws_2, Agws_1, p))$$
$$\wedge (Make\text{-}Offer(Agws_1, Agws_2, p) \wedge C_2 \Rightarrow Challenge(Agws_2, Agws_1, p'))$$
$$\wedge (Make\text{-}Offer(Agws_1, Agws_2, p) \wedge C_3 \Rightarrow Attack(Agws_2, Agws_1, (H, \neg p')))$$
$$\wedge (Make\text{-}Offer(Agws_1, Agws_2, p) \wedge C_4 \Rightarrow Make\text{-}Offer(Agws_2, Agws_1, q))$$
$$\wedge (Make\text{-}offer(Agws_1, Agws_2, p) \wedge C_5 \Rightarrow Refuse(Agws_2, Agws_1, p))$$

where

$C_1 = (p \lhd Arg\_Sys(Agws_2))$

$C_2 = \exists p' \subseteq p: \neg (p' \lhd Arg\_Sys(Agws_2)) \wedge \neg (\neg p' \lhd Arg\_Sys(Agws_2))$

$C_3 = \exists p' \subseteq p: (H \lhd Arg\_Sys(Agws_2)) \wedge (H, \neg p') \mathcal{A} (p,p)$

$C_4 = p \cong q \wedge q \lhd Arg\_Sys(Agws_2)$

$C_5 = \neg(C_1 \vee C_2 \vee C_3 \vee C_4)$

By definition, $Attack(Agws_2, Agws_1, (H, \neg p'))$ means that $Agws_2$ asserts argument $(H, \neg p')$ to attack a part or the whole offer proposed by $Agws_1$. The generation of a set of formulae H from $Agws_2$ is defined as follows:

$$H \lhd Arg\_Sys(Agws_2) \triangleq \forall h_i \in H \ h_i \lhd Arg\_Sys(Agws_2)$$

When an argumentative Web service receives an offer, it accepts it if it has a supporting argument for it, challenges a part of it if it has no argument for or against this part, attacks if it has an argument against the offer, and/or makes a counter offer if it can generate such a counter-offer from its knowledge base using its argumentation system. The content of an attack could also be a counter-offer. If none of these conditions is satisfied, the addressee refuses the offer. For example, an argumentative Web service can refuse an offer if it has already accepted a different offer made by another master Web service about the same subject. Generally, within a dialogue game, an

argumentative Web service can only play one move. However, *Attack* and *Make-Offer* moves can be played together; attacking offers and then making counter-offers.

4.3.3 Challenge and Justification Games

The *Challenge game* is specified as follows:

$Challenge\ (Agws_1, Agws_2, p) \wedge C_1 \Rightarrow Justify(Agws_2, Agws_1, (H, p))$

where: $C_1 = H \lhd \text{Arg\_Sys}(\text{AgWS}_2)$

Condition C_1 should always be satisfied since a Web service must always be able to justify its propositions and assertions.

We specify the *Justification game* as follows:

$(Justify(Agws_1, Agws_2, (H,p)) \wedge C_1 \Rightarrow Accept(Agws_2, Agws_1, H))$

$\wedge (Justify(Agws_1, Agws_2, (H, p)) \wedge C_2 \Rightarrow Challenge(Agws_2, Agws_1, H'))$

$\wedge (Justify(Agws_1, Agws_2, (H, p)) \wedge C_3 \Rightarrow Attack(Agws_2, Agws_1, (H', p')))$

$\wedge (Justify(Agws_1, Agws_2, (H, p)) \wedge C_4 \Rightarrow Make\text{-}Offer(Agws_2, Agws_1, q))$

$\wedge (Justify(Agws_1, Agws_2, (H, p)) \wedge C_5 \Rightarrow Refuse(Agws_2, Agws_1, p))$

where:

$C_1 = H \lhd Arg\_Sys(AgWS_2)$

$C_2 = \exists H' \subseteq H: \forall hi \in H \neg (hi \lhd Arg\_Sys(AgWS_2) \wedge \neg (\neg hi \lhd Arg\_Sys(AgWS_2))$

$C_3 = H' \lhd Arg\_Sys(AgWS_2)) \wedge (H', \neg p') \not\sim (H,p)$

$C_4 = p \cong q \wedge q \lhd Arg\_Sys(AgWS_2)$

$C_5 = \neg(C_1 \vee C_2 \vee C_3 \vee C_4)$

Challenging a set of formulae H means that challenging all the formulas in it:

$Challenge\ (Agws_2, Agws_1, H) \triangleq \forall hi \in H\ Challenge\ (Agws_2, Agws_1, hi)$

These five conditions are similar to those associated with the Offer game. The only difference in the Justification game resides in *Accept* and *Attack* moves that are relative to the support of the offer and not to the offer itself.

4.3.4 Attack Game

The *Attack game* is specified as follows:

$(Attack(Agws_1, Agws_2, (H, p)) \wedge C_1 \Rightarrow Accept(Agws_2, Agws_1, p))$

$\wedge (Attack(Agws_1, Agws_2, (H, p)) \wedge C_2 \Rightarrow Challenge(Agws_2, Agws_1, H'))$

$\wedge (Attack(Agws_1, Agws_2, (H, p)) \wedge C_3 \Rightarrow Attack(Agws_2, Agws_1, (H', p')))$

$\wedge (Attack(Agws_1, Agws_2, (H, p)) \wedge C_4 \Rightarrow Make\text{-}Offer(Agws_2, Agws_1, q))$

$\wedge (Attack(Agws_1, Agws_2, (H, p)) \wedge C_5 \Rightarrow Refuse(Agws_2, Agws_1, p))$

These conditions are identical to the ones associated with the *Justification game*.

An argumentative Web service $Agws_2$ accepts an attacker's argument if it can generate a support for it from its argumentation system. If it cannot generate nor negate this support, the agent challenges it. If it can generate a counter-attacker argument, then it will play the *Attack* move. If an offer can be made from the Web service's knowledge base using its argumentation system, it makes this offer. Otherwise, it refuses the attacker's argument. This *refuse move* can be played if the negation of the attacker's argument conclusion is in $Agws_2$'s knowledge base. We note in this case that $Agws_2$ cannot play the *Attack move* since it does not have a counter-argument but only knowledge about the negation of the argument conclusion.

4.4 Dialogue Games Combination

Having specified the different dialogue games that argumentative Web services use in their interactions to manage their communities, we need to specify how these games could be now combined to form the persuasive negotiation protocol. We notice that during the same protocol session, an argumentative Web service cannot play the same move with the same content more than once. For example, if a master Web service proposed a participation rate by time unit v_2 during a protocol session, the same value cannot be proposed again during this session. Also, if an argumentative Web service uses a counter-argument to attack an argument, it cannot use the same counter-argument afterwards during this session (reiterations are prohibited). The protocol

terminates (*Termination game*) either by accepting or refusing the last offer. There is an acceptance when a Web service accepts the offer (for example accepts the last offered rewards to join the community), i.e., when an agreement is reached. The protocol terminates by a refusal when no agreement is reached. The *P*ersuasive *N*egotiation *P*rotocol for *C*ommunities of *W*eb *S*ervices (*PNP -CWS*) that combines the aforementioned games can be described using the BNF grammar as follows table 4.1:

Table 4.1 BNF grammar for PNP-CWS

PNP -- CWS = Entry game ; (*Refuse*| (*Accept* ; *ChG*))	*Symbols:*
ChG = Make-Offer ; *X*	"|": the choice symbol
X = Accept	";" : the sequence symbol
|*Refuse*	
| *Make-Offer* ; *X*	
| *Attack* ; *X*	
| *Challenge* ; *Justify* ; *X*	
| *Attack* ; *Make-Offer* ; *X*	

After the *Entry game*, the addressee refuses the invitation, or accepts to engage in negotiation, in which case chaining games (*ChG*) will take place. The last line in this grammar refers to the case where *Attack* and *Make-Offer* moves are played together.

4.5 Formal Analysis

4.5.1 *PNP -CWS*'s Properties

Three computational and formal properties of the *PNP-CWS* protocol will be discussed in our research. They are: *termination* (no deadlock), *soundness* (correct specification), and *completeness* (wholeness with respect to Web services' knowledge bases).

Proposition 4.1. *The PNP- CWS protocol terminates iff the invited argumentative Web service refuses the Entry game, or one of the Web services plays either Accept or Refuse moves when the Entry game is accepted.*

Proof. The first part is straightforward, because by definition, if the *Entry game* is refused, the protocol terminates. Let us now suppose that the *Entry game* is accepted. The direction => is straightforward from the protocol's BNF description. In addition let us suppose that the protocol terminates. According to the protocol's BNF description, part *X* is recursive and all the moves, except *Accept* and *Refuse*, are followed by *X*. The only way to stop the process is then to play either *Accept* or *Refuse*. Consequently, the direction <= holds.

Theorem 4.1 (Termination). *For any set of dialogue games, The PNP-CWS protocol always terminates.*

Proof. Because the knowledge bases of argumentative Web services are finite, the arguments that these Web services can build out of these bases are finite as well. Consequently, the numbers of offers and attacks that can be made and built respectively are finite. Therefore, the branches *Make-Offer*; *X*, *Attack*; *X*, and *Attack*; *Make-Offer*; *X* in the BNF description are finite, since playing the same move with the same content is prohibited during a conversation. The branch *Challenge*; *Justify*; *X* is also finite because the number of arguments is finite and when an argument is justified by itself, the addressee cannot challenge it again because repeating moves is prohibited. Thus in all possible executions, one of the argumentative Web services will select one of the branches *Accept* or *Refuse*. From Proposition 4.1 the result follows.

Definition 4.3 (Agreement). *Let Ag_{WS_1} and Ag_{WS_2} be two argumentative Web services engaged in a conversation using the PNP-CWS protocol, and Arg_Sys(Ag_{WS_1}) and Arg _Sys(Ag_{WS_2}) their respective argumentation systems. An agreement about an offer p is reached between Ag_{WS_1} and Ag_{WS_2} iff $p \triangleleft Arg\_Sys(Ag_{WS_1})$ and $p \triangleleft Arg\_Sys(Ag_{WS_2})$.*

In other words, an agreement about an offer is reached iff the offer can be supported by the two agents' argumentation systems of the participating Web services.

Theorem 4.2 (Soundness). *If the PNP-CWS protocol terminates by an acceptance (resp. refusal), then an agreement is (rep. is not) reached.*

Proof. According to the protocol's BNF description, an argumentative Web service plays *Accept* move as a reply to either an offer, an attack, or a justification. According to the *Offer game*, accepting an offer means that the addressee has an argument in its knowledge base that supports accepting this offer. According to Definition 5, having this argument in the knowledge base means that an agreement is reached. Now, if the argumentative Web service accepts to either attack or justify, then according to the *Attack* and *Justification* games, the protocol's BNF description, and the fact that the content of an attack could be a counter-offer, this Web service accepts the last offer made by the addressee. Accepting this offer means that the Web service has an argument supporting it, in the other words, an agreement is reached.

In the opposite case, if an argumentative Web service plays a refusal, then according to the dialogue games specification and the protocol's BNF description, all the exchanged offers can not be supported by one of the two Web services. This means that there is no argument from the two Web services' knowledge bases supporting one of the offers. Consequently, an agreement is not reached.

The soundness property shows that the protocol is correct. However, what is important here to show is that if more than one agreement is made available for argumentative Web services, then the protocol execution will reach one of them.

Theorem 4.3 (Completeness). *If an agreement about an offer p can be reached from the knowledge bases of the argumentative Web services, then the protocol execution will result in achieving an agreement.*

Proof. According to Definition 4.3, the existence of an agreement about p means that $p \triangleleft Arg\_Sys(Ag_{WS_1}) \cup p \triangleleft Arg\_Sys(Ag_{WS_2})$. Then, from the union of the two knowledge bases, it is possible to build an argument supporting the offer p, which is not attacked by another argument from the union.

If p is the initial offer made by $Agws_1$, then $Agws_2$ will accept it since $p \lhd Arg\_Sys(Agws_2)$. So an agreement is reached. If the initial offer is q, we have $q \cong p$ since p and q are different but about the same topic. According to Proposition 4.1, the protocol terminates by either a refusal or an acceptance. Because the protocol always terminates by Theorem 4.1, during the protocol execution one of the argumentative Web services should play either *Refuse* or *Accept* move. Suppose that *Refuse move* is played by one of the two Web services, for example $Agws_1$. According to the dialogue games specification, there is no possibility for this Web service to make a counter-offer r such that $r \cong q \wedge r \lhd Arg\_Sys(Agws_1)$. This is contradictory because by hypothesis there is an offer p such that $p \cong q \wedge p \lhd Arg\_Sys(Agws_1)$. Consequently, the only possibility to terminate the protocol is to play an acceptance move, which means that an agreement is reached.

We notice here that the soundness theorem states that an agreement is reached, but does not tell what the agreement is. The reason is that many agreements can exist, and which one could be reached depends on the strategies that these argumentative Web services adopt.

4.5.2 Complexity Analysis

The *PNP-CWS* protocol is expressed in terms of argumentation-based dialogue games, and, the decision parameters (the conditions associated with the rules) that argumentative Web services use to combine these games are expressed in terms of the possibility of building arguments, the complexity of the protocol is determined by the complexity of generating arguments to support offers or to attack existing arguments. In the following we present the different complexity results.

Proposition 4.2. *Given a Horn knowledge base* Γ, *a subset* $H \subseteq \Gamma$, *and a formula h. Checking whether (H, h) is an argument is polynomial.*

Proof. From the linear time algorithms for Horn satisfy-ability in, it follows that the Horn implication problem $H \vdash h$ is decidable in $O(|H| \times |h|)$ time. From the same result, it also follows that deciding whether H is consistent is polynomial.

Proposition 4.3. *Given a Horn knowledge base* Γ, *and an argument (H, h) over* Γ. *Checking whether (H, h) is minimal is polynomial.*

Proof. Let l be a literal. The following algorithm resolves the problem:

$\forall\ l \in H$ check if $H - \{l\} \vdash h$. Because the implication problem is polynomial, we are done.

As indicated in Section 4.1, argumentative Web services are equipped with knowledge bases that are supposed to be consistent. Let us consider this case.

Proposition 4.4. *Let* Γ *be a definite Horn knowledge base, h a formula, and A the set of arguments over* Γ.

$$\exists H \subseteq \Gamma : (H,h) \in A \Rightarrow \forall H' : H \subseteq H' \subseteq \Gamma, (H',h) \in A .$$

Proof. If (*H, h*) is an argument where H is a set of definite Horn formulas under the form c or $p_1 \wedge p_2 \wedge \ldots \wedge p_n \rightarrow c$ where $p_1, p_2, \ldots\ldots, p_n, c$ are positive literals, then adding any definite Horn formula to H will result in a consistent set of formulas $H' : \Gamma \supseteq H' \supseteq H$. Since $H \vdash h$, it follows that $H' \vdash h$, whence the proposition.

Theorem 4.4. *Given a definite Horn knowledge base* Γ *and a formula h. Deciding whether there is an argument (H, h) is polynomial.*

Proof. From Proposition 4, it follows that there is an argument supporting h iff (Γ, h) $\in$ A. Because every definite Horn knowledge base is a Horn knowledge base, then by Proposition 2, the theorem follows.

The following theorem is a direct consequence of Theorem 4.4.

Theorem 4.5. *Given a consistent Horn knowledge base* Γ *and a formula h. deciding whether there is an argument (H, h) is polynomial.*

Proof. Proposition 4 holds if the knowledge base Γ is consistent. Then, by Proposition 2, the result follows.

Proposition 4.5. *Let Γ be a Horn knowledge base and (H, h) and (H', h') be two arguments over Γ. Deciding whether (H', h') $\not\sim$ (H, h) is polynomial.*

Proof. According to Definition 2, (H', h') $\not\sim$ (H; h) iff $H \vdash \neg h$. The proof is then straightforward since the Horn implication problem $H \vdash \neg h$ is decidable in $O(|H| \times |\neg h|)$ time.

Theorem 4.6. *Let Γ be a consistent Horn knowledge base and (H, h) an argument over Γ. Deciding whether there is an attacker of (H, h) over Γ is polynomial.*

Proof. From Definitions 3.1 and 3.2, building an argument attacking a given argument is less complex than building an argument supporting a conclusion. From Theorem 4.4 we are done.

These results prove that our *PNP-CWS* protocol is computationally efficient, and its complexity depends only on the size of the knowledge bases.

4.6 Implementation

4.6.1 Architecture

A prototype has been implemented to demonstrate first, the combination between argumentative agents and Web services and second, the performance of the persuasive negotiation protocol *PNP-CWS* that manages communities of Web services. The prototype is built on top of Jadex platform. We have introduced in Chapter 2 that based on Java and XML technologies, Jadex allows building up rational and goal-oriented agents. Figure 4.2 depicts the prototype's architecture.

There are two kinds of argumentative Web services agents in the system: Master and Slave. Argumentative Web services have capabilities that represent their beliefs, goals, plans, and events. These capabilities are specified in an XML-based *Definition File*. The description of the

Web service and its non-functional parameters (i.e., QoS) are examples of beliefs. We will split beliefs, goals, and plans to introduce the cases that we simulated for argumentative Web services.

4.6.2 Beliefs

An agent's beliefs are stored in its belief-base. A belief-base is a container that includes believed facts, similar to a simple data-storage, and represents an agent's knowledge about the environment. Unlike most PRS-style BDI systems, Jadex allows to store arbitrary Java objects as beliefs in its belief-base. We store the system time, negotiation recode, agents in the community, etc., values in a agent belief-base. In our case, we use beliefs and belief sets as primary storage capacities for our plans. In this way, a belief or a belief-set in Jadex can retrieve by a declarative OQL-like query language. Furthermore, it allows triggering some action when a fact of a belief set is added or a belief is modified.

Below the program section is picked from our Master Web services agent's ADF. "*Invite*" is a Java class for the structure of invitation. "*agent_list*" is a list of agents in the community. "*negotiation_reports*" records the negotiation procedure.

```
<beliefs>
     ......
     <beliefset name="invites" class="Invite">
           <facts evaluationmode="dynamic">
             select $g.getParameter("invite").getValue()from IRGoal $g
in $goalbase.getGoals("send_invitation")
           </facts>
     </beliefset>

     <belief name="agent_list" class="AgentIdentifier[]">
     </belief>

     <beliefset name="negotiation_reports" class="NegotiationReport"/>

     ......
</beliefs>
```

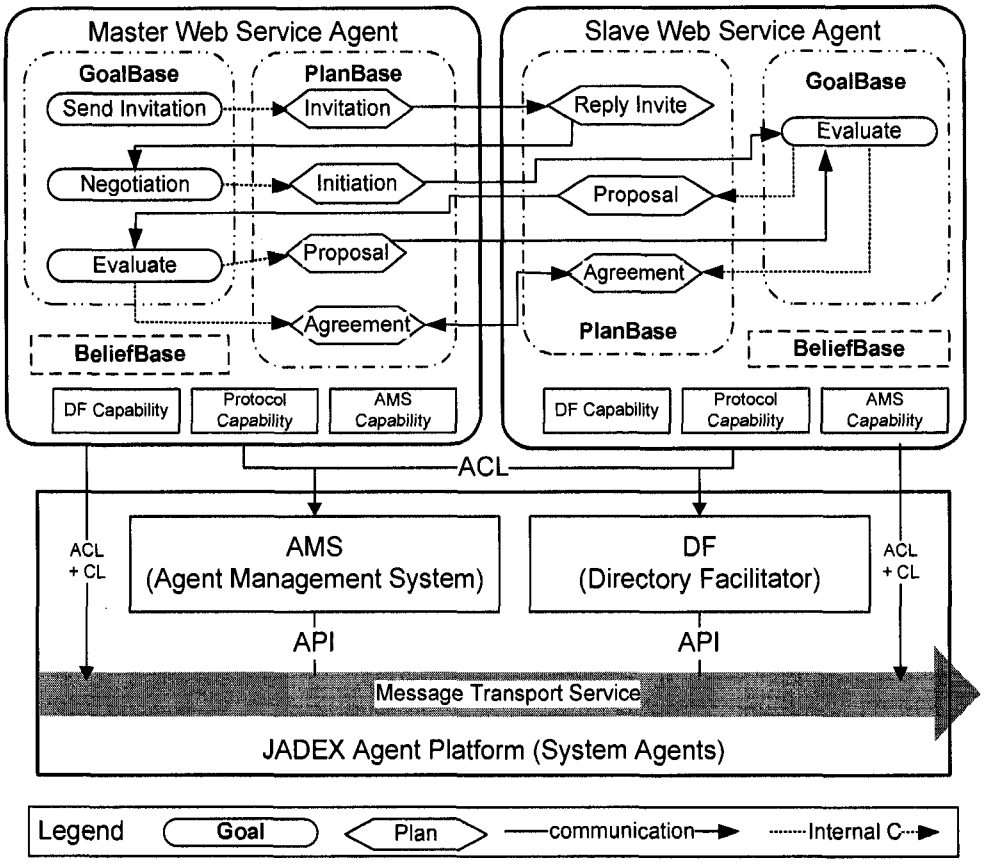

Figure 4.2 Prototype's architecture

4.6.3 Goals

The master Web service has three goals that populate its *goal-base*: (1) *Send Invitation* used to send joining invitations; (2) *Negotiation* used to trigger the negotiation of the joining contracts; and (3) *Evaluate* used to trigger the evaluation of the different offers it receives back.

Each goal has a target condition and is activated when this condition evaluates to true. For example, *Negotiation* goal in the master Web service is activated when an acceptance reply is received by a slave Web service. Two different kinds of goals are involved in our system: *achieve goal* and *query goal*. Achieve goals are used to reach a desired world state. Therefore, most goals in our system are this kind of goal. For example, in agent-based Web-service, the goal *negotiation* is one *achieve goal*, the code is shown below:

```
<goals>
      <achievegoal name="negotiation" recur="true" recurdelay="10000">
            <parameter name="invite" class="Invite">
                  <bindingoptions>$beliefbase.initial_invites
                  </bindingoptions>
            </parameter>
            <unique/>
            <targetcondition>
                  Invite.DONE.equals($goal.invite.getState())
            </targetcondition>
            <failurecondition>
                  $beliefbase.time>$goal.invite.getDeadline().getTime()
            </failurecondition>
      </achievegoal>
</goals>
```

4.6.4 Plans

Four plans in the master Web service are associated to these goals: (1) *Invitation* used to process the joining invitation; (2) *Initiation* triggered by the *Negotiation* goal; (3) *Proposal* to make and evaluate offers; and (4) *Agreement* to stop the negotiation either by an agreement or not. The slave Web service has three plans: (1) *Reply Invitation* used to reply to the joining invitation; (2) *Proposal* to make offers; and (3) *Agreement* to finish the negotiation. *Evaluate* goal in the slave Web service is used to trigger the *Proposal* and *Agreement* plans.

When a plan is triggered, the Web service runs it to perform the specified actions. For example, when *Evaluate* goal is activated, the plan *Proposal* is executed to evaluate the received offer and/or make a new offer. The different parameters of a plan are specified in an ADF file. The following code gives an example of the different parameters used in the *Proposal* plan. For example the file "evaluate proposals.cfp" refers to the mapping goal file of this plan.

```
<goals>
      ......
      <querygoalref name="cnp_evaluate_proposals">
            <concrete ref="procap.cnp_evaluate_proposals"/>
      </querygoalref>
      ......
</goals>

<plans>
      ......
      <plan name="evaluate_proposals_plan">
```

```
            <parameter name="cfp" class="Object">
                  <goalmapping ref="cnp_evaluate_proposals.cfp"/>
            </parameter>
            <parameter name="cfp_info" class="Object" optional="true">
                  <goalmapping ref="cnp_evaluate_proposals.cfp_info"/>
            </parameter>
            <parameterset name="proposals" class="Object">
                  <goalmapping ref="cnp_evaluate_proposals.proposals"/>
            </parameterset>
            <parameterset name="history" class="NegotiationRecord"
                  optional="true">
                  <goalmapping ref="cnp_evaluate_proposals.history"/>
            </parameterset>
            <parameterset name="acceptables" class="Object"
                  direction="out">
                  <goalmapping>
                        ref="cnp_evaluate_proposals.acceptables"
                  <goalmapping/>
            </parameterset>
            <body class="EvaluatePlan" />
            <trigger>
                  <goal ref="cnp_evaluate_proposals"/>
            </trigger>
      </plan>
      ......
</plans>
```

The argumentation reasoning is implemented in Java as the procedural parts of the plans. The *PNP-CWS* protocol is declaratively specified as a set of dialogue games using XML syntax in a public plan that argumentative Web services refer to when communicating. Which dialogue game a Web service should play within this protocol depends on its strategy based on its argumentation system and the message it receives from the addressee. The exchanged messages are events that trigger some plans. Web services share the same ontology to define the meaning of the Horn formulas they exchange. The whole prototype relies on the Jadex platform, which provides a directory facilitator, an agent management system, and a message transport service. The message transport service is provided to transport messages between Web services by specifying the recipient of a message, the general structure of the message, and the used ontology. The agent management system facilitates the creation, registration, location and operation of Web services. All the simulated Web services and communities are registered in the directory facilitator. The directory facilitator used to report Web services in a community is controlled by

the master Web service. When a Web service accepts a joining invitation, the master adds it to its directory. When the master decides to put a Web service out of the community, it deletes its registration information from the directory.

4.7 Experimental Results

To illustrates some experimental results, we consider two scenarios: (1) invitation to join a community; and (2) negotiation of the joining contract. In the first scenario, a Master Web service *MWS* searches for slave Web services providing Weather-Forecast functionality in the directory facilitator. Five Slave Web Services are identified $SWS_{i;i=1;:::;5}$, and the master plays the *Entry game* (*Open* communicative act) to invite them to negotiate about joining its community. Three of these Slave Web services (SWS_1, SWS_2, and SWS_5) accept the game because they have arguments supporting the acceptance decision. These arguments are availability (these Web services did not commit to join any other community yet) and interest in joining the community of this master Web service because they believe that its reputation is high. The rest of the slave Web services namely SWS_3 and SWS_4 refuse the invitation for two different reasons. SWS_3 is already committed to join another community, and SWS_4 is not interested because it has an argument against the reputation of this community. Figure 4.4 illustrates the sequence diagram of this scenario. According to the *Entry game*, slave Web services do not reveal their arguments about their acceptance or refusal, but they use them internally to make their decisions. This is the reason why they are not illustrated in the sequence diagram. Fig. 6 illustrates a snapshot of this scenario.

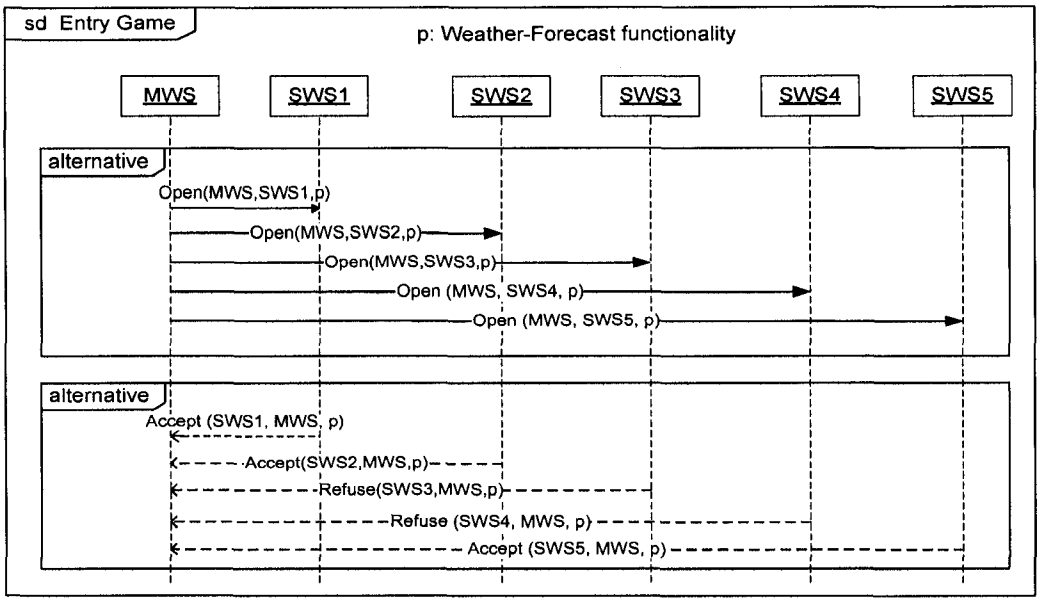

Figure 4.3. Sequence diagram for Entry game

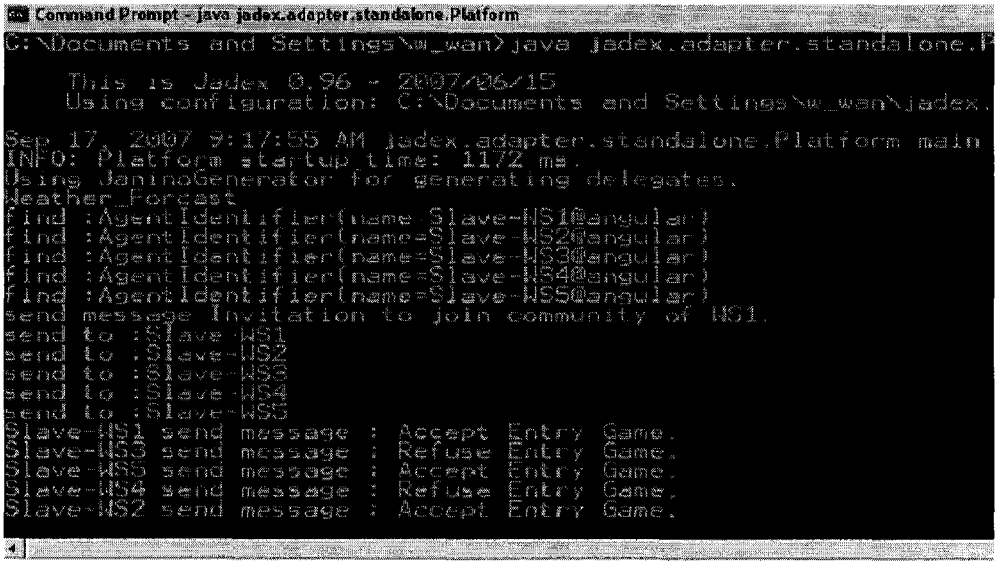

Figure 4.4 Snapshot for the *Entry game* scenario

In the second scenario, the master Web service negotiates the joining contract with one of the slave Web services that accept the joining invitation, for instance SWS_1. Two terms are negotiated in this scenario: SWS_1's *availability* increase and *participation rate* in composite business scenarios the master Web service can provide. First, the master Web service uses its argumentation system to make an offer: promised participation rate by time unit = 20% and availability increase by time unit = 30% (these values are calculated on the basis of the current

efficiency of the community and what the master provides to the existing members). By playing the *Offer game*, *SWS*1 attacks the first part of the offer because it has an argument against this offer; the proposed participation rate is less than the community average (we call this argument the average argument). The master Web service replies, using the *Attack game*. It makes a new offer in which the participation rate is increased (from 20% to 30%) and the availability is decreased (from 30% to 25%). Playing the *Offer game*, *SWS*1 replies to this new offer by attacking the second part using the argument that adding its current availability to the offered rate is less than the community average. The master makes then a new offer by increasing the availability rate (from 25% to 35%) but by reducing the participation rate (from 30% to 25%). This new offer considers the fact that it is greater than the community average, so it cannot be attacked using the average argument. *SWS*1 replies to this new offer by making another counter-offer using its argumentation system. In this offer, *SWS*1 requests a 28% for the participation rate instead of 25% and accepts the proposed availability increase. The master Web service proposes then a balance between the two values: 27% for the participation rate, and 32% for the availability increase. Finally, *SWS*1 accepts the new offer, which results in stopping the negotiation. This scenario is illustrated with Figure 4.5.

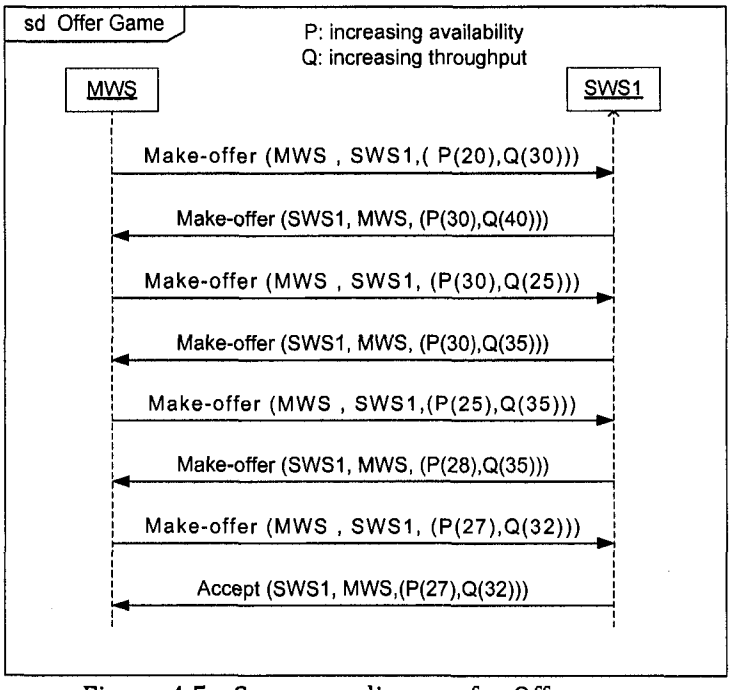

Figure 4.5. Sequence diagram for Offer game

4.8 Conclusion

In this chapter, we discussed the rationale behind using argumentation to manage Web services communities, and we showed that this technique provides suitable solutions for autonomous Web services. We framed the management operations that take place in a community with a persuasive negotiation protocol that argumentative agents implement. This protocol relies on a set of games like *Entry*, *Offer*, and *Attack*. The formal properties of this protocol along with its complexity were discussed and assessed, respectively. In addition, a prototype simulating the protocol was discussed. The experimental results revealed that inviting Web services to join communities and negotiating joining contracts with multi-issue terms can be efficiently managed using argumentation. In addition, the implementation allowed us to experimentally check the theoretical soundness and completeness of the proposed protocol.

Chapter 5. Verifying Communities of Web Services

This chapter gives an overview of our research work on an approach to verify Communities of Web Services with formal methods, which offer a potential to obtain an early integration of verification in the design process. First, we abstract the communities of Web services as a model. Furthermore, we translate this model to Process Algebra Calculus of Communicating Systems (CCS). Then, we verify this model using a modified and enhanced version of Concurrency Workbench of New Century (CWB-NC). The content of this chapter is based on our paper [3].

The structure of the chapter is organized as follow. In Section 5.1, we overview some model checking techniques. In Section 5.2, we introduce the CTL^{*CA} logic for communicating agents that we use to specify the properties to be checked. In Section 5.3, we define a protocol for communicating agents. We discuss the model checking technique of this protocol using tableau rules in Section 5.4. In Section 5.5, we prove the termination property of the technique and we discuss its computational complexity. We also provide a case study for this approach along with its implementation in Section 5.6. Finally, we conclude the chapter in Section 5.7.

5.1 Overview

Model checking is a problem solving technique establishing whether or not a given formula φ is true in a given model M. Therefore, model checking has two input parameters: the formula φ and the model M. Generally speaking, model checking enables the formal verification of a variety of specification patterns, which usually using temporal logic formulae, in distributed systems. Using model checking, the problem of verifying that a generic system S complies with a

given specification P is reduced to the problem of verifying that a logical formula φ_P (representing the specification P) is satisfied in a model M_S (representing the generic system S).

There are two main formal verification approaches: proof-based approaches and model-based approaches. In the proof-based approaches, the system description is a set of logical formulae Γ and the specification is another formula ϕ. The verification method consists of trying to find a proof that $\Gamma \vdash \phi$. This typically requires guidance and expertise from the user in order to identify suitable lemmas and auxiliary assertions. In the model-based approaches, also called *model checking*, the system (the protocol) is represented by a finite model M modeled as a *Kripke structure* using an appropriate logic. The specification is again represented by a formula ϕ expressed in the same logic, and the verification method consists of computing whether the model M satisfies ϕ or not ($M \models \phi$). This is an algorithmic-based technique and usually done automatically.

The model checking approach we use is based on an alternative view of model checking proposed by Bhat and Cleaveland [11] and Bhat et al [12]. This view relies on translating formulae into intermediate structures, *Alternating Büchi Tableau Automata* (ABTA). Unlike the other model checking techniques, this technique allows us to verify not only temporal formulae, but also action formulae. Because our logic for communicating agents is based on an action theory, this technique is more suitable. This approach is called *tableau-based model checking* [4].

Tableau-based model checking, which is a *top-down or goal-oriented* approach, is based on the use of assertions and tableau rules which are proof rules. Assertions are typically of the form $s \vdash M \phi$ and mean that state s in Model M satisfies the formula ϕ. Using a set of tableau rules we aim to prove the truth or falsity of assertions. According to this approach, we start from a goal, and we apply a proof rule and determine the sub-goals to be proven. The proof rules are designed so that the goal is true if all the sub-goals are true.

We propose a model checking-based verification of dialogue game protocols using a temporal and dynamic logic in our research. The dynamic aspect of our logic is represented by action formulae and not by strengthening until operator by indexing it with the regular programs of dynamic logic. Our protocols are specified as actions that agents apply to propositional commitments (PC). In addition, the model checking procedure that we propose allows us to verify not only that the dialogue game protocol (the theoretical model) satisfies a given property, but also that the tableau semantics of the communicative acts is respected. The idea is to integrate this semantics in the specification of the protocol, and then to propose a parsing method to verify that the protocol specification respects the semantic definition. Consequently, if agents respect these protocols, then they also respect the semantics of the communicative acts. We have here a mechanism for checking the agents' compliance with the semantics without taking into account the agents' specifications created by the developers. Indeed, we have only one procedure to verify: 1) the correctness of the protocols relative to the properties that the protocols should satisfy; and 2) the conformance of agents to the semantics of the communicative acts. The purpose of this technique is to verify the temporal properties of the protocol and to ensure that the structures of the communicative acts are the same in both the protocol and the specification.

5.2 CTL^{*CA} Logic for Communities of Web Services

5.2.1 Syntax

CTL^{*CA} is an extended logic from CTL* by adding propositional commitments and action formulae for Communicative Agent. The syntax of this logic is shown below table 5.1

In this thesis we use $p, p_1, p_2 \ldots$ to range over the set of atomic propositions Φ_p. There are two kinds of formulae in the logic: they are state formulae, which are generated by S , and path formula, which are generated by P . We use $\psi, \psi_1, \psi_2, \ldots$ to range over state formulae and

$\phi, \phi_1, \phi_2, \ldots$ to range over path formulae. The meaning of most of the constructs is straightforward (from CTL* with next (X^+), previous (X^-), until (U^+), and since (U^-) operators). The formula $\phi_1 \therefore \phi_2$ means that ϕ_1 is an argument for ϕ_2. We can read this formula: ϕ_1 so ϕ_2. This operator introduces argumentation as a logical relation between path formulae.

Table 5.1 The Syntax of CTL*CA Logic

$\mathrm{P} ::= \mathrm{S} \mid \mathrm{P} \wedge \mathrm{P} \mid \mathrm{P} \vee \mathrm{P} \mid X^+\mathrm{P} \mid X^-\mathrm{P} \mid \mathrm{P}\, U^+\, \mathrm{P} \mid \mathrm{P}\, U^-\, \mathrm{P} \mid \mathrm{P}\, \therefore\, \mathrm{P}$
$\mid PC(Ag_1, Ag_2, t, \mathrm{P})$
$\mid C(Ag_1, PC(Ag_1, Ag_2, t, \mathrm{P}))$
$\mid Act(Ag_i, PC(Ag_1, Ag_2, t, \mathrm{P}))$

The formula $PC(Ag_1, Ag_2, t, \mathrm{P})$ is the propositional commitment made by agent Ag_1 at the moment t towards agent Ag_2 that the path formula P is true. The formula $C(Ag_1, PC(Ag_1, Ag_2, t, \mathrm{P}))$ means that agent Ag_1 creates the commitment $PC(Ag_1, Ag_2, t, \mathrm{P})$. $Act(Ag_i, PC(Ag_1, Ag_2, t, \mathrm{P}))$ means that agent Ag_i ($i \in \{1, 2\}$) performs an action on the propositional commitment made by Ag_1 towards Ag_2. The set of actions performed on propositional commitments are *Challenge*, *Accept*, *Refuse*, *Justify*, *Attack*, *Defend*. More details information about these actions will be introduced later.

5.2.2 Semantics

The formal model M associated to this logic corresponds to the agent communication protocol. Formally, this model is defined as: $M = \langle S_m, Lab_m, Act_m, \xrightarrow{Act_m}, Agt, R_{PC}, S_{m_0} \rangle$ where: S_m is a set of states; $Lab_m : S_m \to 2^{\Phi_P}$ is the labeling state function; Act_m is the set of actions performed on propositional commitments; $\xrightarrow{Act_m} \subseteq S_m \times Act_m \times S_m$ is the transition relation; Agt is a set of agents; $R_{PC} : S_m \times Agt \times Agt \to 2^{\sigma}$ with σ is the set of all paths in M, is an accessibility

modal relation that associates to a state s_m the set of paths representing the propositional commitment along which an agent can commit towards another agent; s_{m_0} is the start state. The paths that path formulae are interpreted over have the form $x^i = s_{m_i} \xrightarrow{\alpha_{i+1}} s_{m_{i+1}} \xrightarrow{\alpha_{i+2}} s_{m_{i+2}} \ldots$ where $x^i \in \sigma$, $s_{m_i}, s_{m_{i+1}}, \ldots$ are states and $\alpha_{i+1}, \alpha_{i+2}, \ldots$ are actions.

The semantics of CTL*^{CA} state formulae is as usual (semantics of CTL*). A path satisfies a state formula if the initial state in the path does. Along a path x^i, $\phi_1 \therefore \phi_2$ holds if ϕ_1 is true and at next time if ϕ_1 is true then ϕ_2 is true. Formally:

$$x^i \text{ ‘}_M \phi_1 \therefore \phi_2 \text{ iff } x^i \text{ ‘}_M \phi_1 \,\&\, x^{i+1} \text{ ‘}_M \phi_1 \Rightarrow \phi_2$$

A path x^i satisfies $PC(Ag_1, Ag_2, t, \phi)$ if every accessible path to Ag_1 towards Ag_2 from the first state of the path using R_{PC} satisfies ϕ. formally:

$$x^i \text{ ‘}_M PC(Ag_1, Ag_2, t, \phi)) \text{ iff}$$
$$\forall x^j \in \sigma,\ x^j \in R_{PC}(s_{m_i}, Ag_1, Ag_2) \Rightarrow x^j \text{ ‘}_M \phi$$

A path x^i satisfies $C(Ag_1, PC(Ag_1, Ag_2, t, \phi))$ if C is in the label of the first transition on this path and $PC(Ag_1, Ag_2, t, \phi)$ holds along the path x^{i+1}. formally:

$$x^i \text{ ‘}_M C(Ag_1, PC(Ag_1, Ag_2, t, \phi)) \text{ iff}$$
$$\alpha_{i+1} = C \,\&\, x^{i+1} \text{ ‘}_M PC(Ag_1, Ag_2, t, \phi)$$

A path x^i satisfies $Act(Ag_i, PC(Ag_1, Ag_2, t, \phi))$ if Act is in the label of the first transition on this path and if in the past (P) Ag_1 has already created the social commitment. Formally:

$$x^i \text{ ‘}_M Act(Ag_1, PC(Ag_1, Ag_2, t, \phi)) \text{ iff } \alpha_{i+1} = Act \,\&\, P(C(Ag_1, PC(Ag_1, Ag_2, t, \phi)))$$

We notice that the past (P) and future (F) operators are abbreviations from until operator (U) in the usual way of CTL* logic.

The tableau rules for CTL*^{CA} are given in Appendix I: Tables AppI.1, AppI.2, AppI.3, and AppI.4.

5.3 Dialogue Game Protocol for Communicating Agents

5.3.1 Dialogue Game Protocols

The dialogue game protocols are specified as a set of rules describing the entry condition, the dynamics and the exit condition of the protocol [7]. These rules can be specified in the logic for communicating agents as action formulae (actions on propositional commitments). We define these protocols as transition systems that are labeled with communicative acts. Such acts are modeled as actions performed by agents *on Propositional Commitments* (*PC*), for example, creating, accepting, or challenging a propositional commitment [9]. The purpose of these transition systems is to describe not only the sequence of the allowed actions (classical transition systems), but also the structure of these actions. The states of these transition systems are sub-transition systems (called *structure transition systems*) describing the structure of the actions labeling the entry transitions. Defining transition systems in such a way allows for the verification of: 1) the correctness of the protocol (if the model of the protocol satisfies the properties that the protocol should specify); 2) the compliance to the structure of the communicative actions (if the specification of the protocol respects the structure).

The definition of the transition system of dialogue game protocols is given by the definition 5.1 and definition 5.2:

Definition 5.1 *A structure transition system T' describing the structure of an action formula is a 7-tuple <S', Lab', F, Ls', R, →, s'$_0$> where:*

- *S' is a set of states,*
- *Lab' : S' → 2^{Φ_p} is the labeling state function, where Φ_p is the set of atomic propositions,*
- *F is a sub-set of CTL*^{CA} formulae (F does not include the action formulae i.e. Satisfy, Accept, etc.),*
- *Ls' : S' → F is a function associating to each state a formula,*
- *$R \in \{ \wedge, \vee, \neg, ?, <\equiv>, X^+, X^-, PC_{Ag} \}$ is the set of tableau rule labels (without the rules for action formulae),*
- *$\rightarrow \subseteq S' \times R \times S'$ is the transition relation,*
- *s'$_0$ is the start state.*

Intuitively, states s' contain the sub-formulae of the action formulae, and the transitions are labeled by operators associated with the formula of the starting state. Semantic transition systems enable us to describe the semantics of formulae by sub-formulae connected by logical operators. Thus, there is a transition between states s'_i and s'_j iff $L'(s'_j)$ is a sub-formula or an semantically equivalent formula of $L'(s'_i)$. Following traditional usage we write $s \rightarrow^r s'$ instead of $<s, r, s'> \in \rightarrow$ where $s, s' \in S'$ and $r \in R$.

Definition 5.2 *A transition system T for a dialogue game protocol is a 7-tuple* $<S, Lab, \wp, L, Act, \rightarrow, s_0>$ *where*:

- S *is a set of states,*
- $Lab : S \rightarrow 2^{\Phi p}$ *is the labeling state function,*
- $\wp$ *is a set of structure transition systems with* $\varepsilon \in \wp$ *is the empty semantic transition system,*
- $L : S \rightarrow \wp$ *is the function associating to a state* $s \in S$ *a semantic transition system* $T' \in \wp$ *describing the semantics of the action labeling the entry transition,*
- $Act \in \{C, Withdraw, Satisfy, Accept, Refuse, Challenge, Justify, Defend, Attack\}$ *is the set of actions,*
- $\rightarrow \subseteq S \times Act \times S$ *is the transition relation,*
- s_0 *is the start state with* $L(s_0) = \varepsilon$ *(i.e. there is no structure transition system in* s_0*).*

The transitions are labeled by the actions applied to propositional commitments. We write $s \rightarrow s'$ instead of $<s, \bullet, s'> \in \rightarrow$ where $s, s' \in S$ and $\bullet \in Act$. Figure 5.1 illustrates a part of a transition system for a dialogue game protocol.

5.3.2 Dialogue Game Protocol Properties

The properties to be verified in the dialogue game protocols specified in CTL^{*CA} are action and temporal properties. For example, we can verify if a model of dialogue game protocol satisfies the following property:

$$AG^{+}(Challenge(Ag_2, PC(Ag_1, Ag_2, t, \phi)) \Rightarrow F^{+}Justify(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi)))$$

This property says that in all paths (A), globally (G^{+}), if an agent Ag_2 challenges the content ϕ of an Ag_1's propositional commitment (PC), then in the future (F^{+}) Ag_1 will justify this content by an argument $\phi' \therefore \phi$.

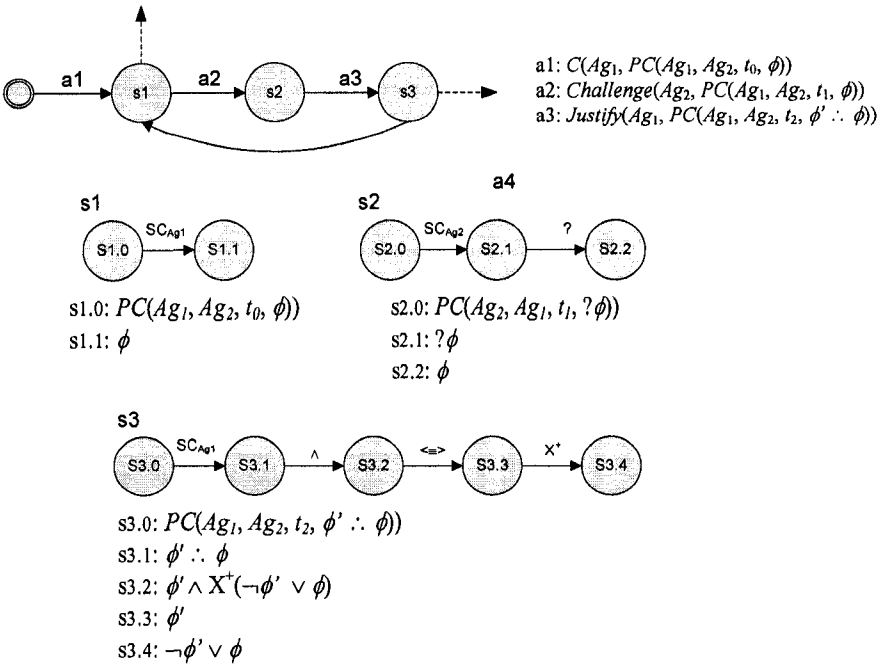

Figure 5.1 A part of a transition system for a dialogue game protocol

Another interesting property to be checked in dialogue games is related to the communicative acts an agent is allowed to perform at a given moment. For example, it is prohibited to attack commitment content if the addressee did not commit about this content. This property is specified using the past operator F^- as follows:

$$AG^+(Attack(Ag_2, PC(Ag_1, Ag_2, t, \phi)) \Rightarrow F^-C(Ag_1, PC(Ag_1, Ag_2, t, \phi)))$$

A third property capturing the deontic notion of propositional commitments is given by the following formula:

$$\begin{aligned} AG^+(&Attack(Ag_2, PC(Ag_1, Ag_2, t, \phi' \therefore \neg \phi)) \Rightarrow \\ &(F^+Defend(Ag_1, PC(Ag_1, Ag_2, t, \phi'' \therefore \phi)) \\ &\vee F^+Attack(Ag_1, PC(Ag_2, Ag_1, t', \phi'' \therefore \neg \phi')) \\ &\vee F^+Accept(Ag_1, PC(Ag_2, Ag_1, t', \phi'))))) \end{aligned}$$

Using this property, we can verify if a model of a dialogue game protocol satisfies the fact that if an agent Ag_2 attacks the content of an agent Ag_1's propositional commitment PC, then Ag_1 will defend its propositional commitment content, attack the Ag_2's argument or accept it.

5.4 Model Checking Technique

We use three sections to describe our combination of an automata-theoretic approach and a tableau-based approach to model-checking Communities of Web service system.

5.4.1 Alternating Büchi Tableau Automata for CTL*^{CA}

Alternating Büchi Tableau Automates (ABTAs) [12] are used to prove properties of *infinite* behavior. These automata can be used as an intermediate representation for system properties. Let Φ_p be the set of atomic propositions and let $\Re$ be a set of tableau rule labels defined as follows:

$\Re = \{\wedge, \vee, \neg, ?\} \cup \Re_{Act} \cup \Re_{\neg Act} \cup \Re_{SC} \cup \Re_{Set}$ where $\Re_{Act}$, $\Re_{SC}$ and $\Re_{Set}$ are defined as follows:

$\Re_{Act} = \{<C>, <W>, <S^{Ag}_{PC}>, <V^{Ag}_{PC}>, <Rea>, <Ch>, <Acc>, <Ref>, <Jus>, <Att>, <Off>\}$.

$\Re_{SC} = \{[PC_{Ag}]\}$.

$\Re_{Set} = \{<\equiv>, X^+, X^-\}$.
The associated tableau rules are given in Appendix. I

Formally, we define ABTAs for CTL*^{CA} logic as follows:

Definition 5.3 *An ABTA for CTL*^{CA} is a 5-tuple $<Q, l, \rightarrow, q_0, F>$, where:*

- *Q is a finite set of states,*
- *$l: Q \rightarrow \Phi_p \cup \Re$ is the state labeling,*
- *$\rightarrow \subseteq Q \times Q$ is the transition relation,*
- *q_0 is the start state,*
- *$F \subseteq 2^Q$ is the acceptance condition.*

ABTAs encode temporal formulae in a "*top-down*" fashion. Indeed, an ABTA uses a proof schema to prove in a goal-directed manner in which a transition system satisfies a temporal formula.

Here is an example in order to understand. We would like to prove that a state s in a transition system satisfies a temporal formula of the form F1 $\wedge$ F2, where F1 and F2 are two formulae. Regardless of the structure of the system, there would be two sub-goals if we want to prove this in a top-down, goal-directed manner. The first would be to prove that s satisfies F1, and the second would be to prove that s satisfies F2. Intuitively, an ABTA for F1 $\wedge$ F2 would encode this "proof structure" using states for the formulae F1 $\wedge$ F2, F1, and F2. A transition from F1 $\wedge$ F2 to each of F1 and F2 should be added to the ABTA and the labeling of the state for F1 $\wedge$ F2 being "$\wedge$" which is the label of a certain rule. Indeed, in an ABTA, we can consider that:

1) states correspond to "formulae",
2) the labeling of a state is the "logical operator" used to construct the formula, and
3) the transition relation represents a "sub-goal" relationship.

In order to decide about the satisfaction of formulae, we use the notion of the *accepting runs* of an ABTA on a transition system. These runs are not considered to be finite, but rather infinite, while cycling infinitely many times through acceptance states. To define this notion of the ABTA's run, we need to introduce three types of nodes: *positive*, *negative* and *neutral* (neither positive nor negative). Intuitively, nodes classified positive are nodes that correspond to a formula without negation (for example $C(Ag_1, PC(Ag_1, Ag_2, t, \phi))$), and negative nodes are nodes that correspond to a formula with negation (for example $\neg Justify(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))$). Neutral nodes are used in order to verify the semantics of an action formula ($act \in Act$) written in the formula to be verified under the form $\neg act$. From the syntax point of view, $\neg act$ means that the action *act* is not performed. For example, if in the formula to be verified appears the sub-formula: $\neg Justify(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))$, we use in the ABTA neutral nodes in order to verify

the semantics of: $Justify(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))$. The reason is that in transition systems, and consequently in the sub-transition systems, we have only action formulae without negation, whereas in the formula to be verified, we can have action formulae with negation. We note that we cannot use here negative nodes because we do not interested in the formula in itself (i.e. in the example $\neg Justify(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))$) but in the semantics of the underlying action (i.e. $Justify(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))$). In other words, we are not interested in the semantics of the negation action, but in the semantics of the action itself. We note here that in order to verify that an action formula $\neg act$ is satisfied, we have to verify that from a given state there is no transition in the transition system labeled by *act*. Definition 5.4 gives the definition of this notion of run. In this definition, elements of the set S of states are denoted s_i or t_i. The explanation of the different clauses is given after the definition.

Definition 5.4 *A run of an ABTA* $B = \langle Q, l, \rightarrow, q_o, F \rangle$ *on a transition system* $T = \langle S, Lab, \wp, L, Act, \rightarrow, s_0 \rangle$ *is a graph in which the nodes are classified as positive, negative, or neutral and are labeled by elements of* $Q \times S$ *as follows*:

1. *The root of the graph is a positive node and is labeled by* $\langle q_0, s_0 \rangle$.
2. *If* φ *is a positive node with label* $\langle q, s_i \rangle$ *such that* $l(q) = \neg$ *and* $q \rightarrow q'$, *then* φ *has one negative successor labeled* $\langle q', s_i \rangle$ *and vice versa.*

- *Otherwise, for a positive node* φ *labeled by* $\langle q, s_i \rangle$:

3. *If* $l(q) \in \Phi_p$ *then* φ *is a leaf.*
4. *If* $l(q) \in \{\wedge, \langle\equiv\rangle\}$ *and* $\{q' \mid q \rightarrow q'\} = \{q_1, \ldots, q_m\}$, *then* φ *has positive successors* $\varphi_1, \ldots, \varphi_m$ *with* φ_j *labeled by* $\langle q_j, s_i \rangle$ $(1 \leq j \leq m)$.
5. *If* $l(q) = \vee$ *then* φ *has one positive successor* φ' *labeled by* $\langle q', s_i \rangle$ *for some* $q' \in \{q' \mid q \rightarrow q'\}$.
6. *If* $l(q) = X^+$ *and* $q \rightarrow q'$ *and* $\{s' \mid s_i \rightarrow^{\bullet} s'\} = \{t_1, \ldots, t_m\}$ *where* $\bullet \in Act$, *then* φ *has positive successors* $\varphi_1, \ldots, \varphi_m$ *with* φ_j *labeled by* $\langle q', t_j \rangle$ $(1 \leq j \leq m)$.
7. *If* $l(q) = X^-$ *and* $q \rightarrow q'$ *and* $\{s' \mid s' \rightarrow^{\bullet} s_i\} = \{t_1, \ldots, t_m\}$ *where* $\bullet \in Act$, *then* φ *has positive successors* $\varphi_1, \ldots, \varphi_m$ *with* φ_j *labeled by* $\langle q', t_j \rangle$ $(1 \leq j \leq m)$.

8. *If* $l(q) = \langle\bullet\rangle$ *where* $\bullet \in Act$ *and* $q \rightarrow q'$, *and* $s_i \rightarrow^{\bullet} s_{i+1}$ *then* φ *has one positive successor* φ' *labeled by* $\langle q', s_{i+1,0}\rangle$ *where* $s_{i+1,0}$ *is the initial state of the semantic transition system of* s_{i+1}.
9. *If* $l(q) = \langle\bullet\rangle$ *where* $\bullet \in \neg Act$ *and* $q \rightarrow q'$, *and* $s_i \rightarrow^{\neg\bullet} s_{i+1}$ *then* φ *has one neutral successor* φ' *labeled by* $\langle q', s_{i+1,0}\rangle$ *where* $s_{i+1,0}$ *is the initial state of the semantic transition system of* s_{i+1}.
10. *If* $l(q) = \langle\bullet\rangle$ *where* $\bullet \in \neg Act$ *and* $q \rightarrow q'$, *and* $s_i \rightarrow^{\bullet'} s_{i+1}$ *where* $\bullet \neq \bullet'$ *and* $\bullet' \in Act$, *then* φ *has one positive successor* φ' *labeled by* $\langle q', s_{i+1}\rangle$.

- *Otherwise, for a negative node* φ *labeled by* $\langle q, s_i\rangle$:

11. *If* $l(q) \in \Phi_p$ *then* φ *is a leaf.*
12. *If* $l(q) \in \{\vee, \langle\equiv\rangle\}$ *and* $\{q' \mid q \rightarrow q'\} = \{q_1, \ldots, q_m\}$, *then* φ *has negative successors* $\varphi_1, \ldots, \varphi_m$ *with* φ_j *labeled by* $\langle q_j, s_i\rangle$ $(1 \leq j \leq m)$.
13. *If* $l(q) = \wedge$ *then* φ *has one negative successor* φ' *labeled by* $\langle q', s_i\rangle$ *for some* $q' \in \{q' \mid q \rightarrow q'\}$.
14. *If* $l(q) = X^{+}$ *and* $q \rightarrow q'$ *and* $\{s' \mid s_i \rightarrow^{\bullet} s'\} = \{t_1, \ldots, t_m\}$ *where* $\bullet \in Act$, *then* φ *has negative successors* $\varphi_1, \ldots, \varphi m$ *with* φ_j *labeled by* $\langle q', t_j\rangle$ $(1 \leq j \leq m)$.
15. *If* $l(q) = X^{-}$ *and* $q \rightarrow q'$ *and* $\{s' \mid s' \rightarrow^{\bullet} s_i\} = \{t_1, \ldots, t_m\}$ *where* $\bullet \in Act$, *then* φ *has negative successors* $\varphi_1, \ldots, \varphi_m$ *with* φ_j *labeled by* $\langle q', t_j\rangle$ $(1 \leq j \leq m)$.
16. *If* $l(q) = \langle\bullet\rangle$ *where* $\bullet \in Act$ *and* $q \rightarrow q'$, *and* $s_i \rightarrow^{\bullet} s_{i+1}$ *then* φ *has one negative successor* φ' *labeled by* $\langle q', s_{i+1,0}\rangle$ *where* $s_{i+1,0}$ *is the initial state of the semantic transition system of* s_{i+1}.
17. *If* $l(q) = \langle\bullet\rangle$ *where* $\bullet \in \neg Act$ *and* $q \rightarrow q'$, *and* $s_i \rightarrow^{\neg\bullet} s_{i+1}$ *then* φ *has one neutral successor* φ' *labeled by* $\langle q', s_{i+1,0}\rangle$ *where* $s_{i+1,0}$ *is the initial state of the semantic transition system of* s_{i+1}.
18. *If* $l(q) = \langle\bullet\rangle$ *where* $\bullet \in \neg Act$ *and* $q \rightarrow q'$, *and* $s_i \rightarrow^{\bullet'} s_{i+1}$ *where* $\bullet \neq \bullet'$ *and* $\bullet' \in Act$, *then* φ *has one negative successor* φ' *labeled by* $\langle q', s_{i+1}\rangle$.

- *Otherwise, for a neutral node* φ *labeled by* $\langle q, s_{ij}\rangle$:

19. *If* $l(q) = \langle\equiv\rangle$ *and* $\{q' \mid q \rightarrow q'\} = \{q_1, q_2\}$ *such that* q_1 *is a leaf, and* s_{ij} *has a successor* s_{ij+1}, *then* φ *has one positive leaf successor* φ' *labeled by* $\langle q_1, s_{ij}\rangle$ *and one neutral successor* φ'' *labeled by* $\langle q_2, s_{i,j+1}\rangle$.
20. *If* $l(q) = \langle\equiv\rangle$ *and* $\{q' \mid q \rightarrow q'\} = \{q_1, q_2\}$ *such that* q_1 *is a leaf, and* s_{ij} *has no successor, then* φ *has one positive leaf successor labeled by* $\langle q_1, s_{ij}\rangle$.

- *Otherwise, for a positive (negative) node* φ *labeled by* $\langle q, s_{ij}\rangle$:

21. *If* $l(q) = \langle\equiv\rangle$ *and* $\{q' \mid q \rightarrow q'\} = \{q_1, q_2\}$ *such that* q_1 *is a leaf, and* s_{ij} *has a successor* s_{ij+1}, *then* φ *has one positive leaf successor*φ' *labeled by* $\langle q_1, s_{ij}\rangle$ *and one positive (negative) successor* φ'' *labeled by* $\langle q_2, s_{i,j+1}\rangle$.

22. *If* $l(q) = \langle\equiv\rangle$ *and* $\{q' \mid q \rightarrow q'\} = \{q_1, q_2\}$ *such that* q_1 *is a leaf, and* $s_{i,j}$ *has no successor, then* φ *has one positive leaf successor* φ' *labeled by* $\langle q_1, s_{i,j}\rangle$ *and one positive (negative) successor* φ'' *labeled by* $\langle q_2, s_i\rangle$.

- *Otherwise, for a positive (negative, neutral) node* φ *labeled by* $\langle q, s_{i,j}\rangle$:

23. *If* $l(q) \in \{\wedge, \vee, ?, X^+, X^-, [PC_{Ag}]\}$ *and* $\{q' \mid q \rightarrow q'\} = \{q_1\}$, *and* $s_{i,j} \rightarrow^r s_{i,j+1}$ *such that* $r = l(q)$, *then* φ *has one positive (negative, neutral) successor* φ' *labeled by* $\langle q_1, s_{i,j+1}\rangle$.

The notion of run of an ABTA on a transition system is a non-synchronized product graph of the ABTA and the transition system. This run uses the label of nodes in the ABTA ($l(q)$), the transitions in the ABTA ($q \rightarrow q'$), and the transitions in the transition system ($s_i \rightarrow s_j$). The product is not synchronized in the sense that it is possible to use transitions in the ABTA while staying in the same state in the transition system (this is the case for example of the clauses 2, 4, and 5).

The second clause in the definition says that if we have a positive node φ in the product graph such that the corresponding state in the ABTA is labelled with $\neg$ and we have a transition $q \rightarrow q'$ in this ABTA, then φ has one negative successor labelled with $\langle q', s_i\rangle$. In this case we use a transition from the ABTA and we stay in the same state of the transition system. In the case of a positive node and if the current state of the ABTA is labelled with $\wedge$, all the transitions of this current state of the ABTA are used (clause 4). However, if the current state of the ABTA is labelled with $\vee$, only one arbitrary transition from the ABTA is used (clause 5). The intuitive idea is that in the case of $\wedge$, all the sub-formulae must be true in order to decide about the formula of the current node of the ABTA, and in the case of $\vee$ only one sub-formula must be true.

The cases in which a transition of the transition system is used are:

1. The current node of the ABTA is labelled with X^+ (which means a next state in the transition system) or X^- (which means a previous state in the transition system). This is

the case of the clauses 6, 7, 14, and 15. In this case we use all the transitions from the current state s_i to next or previous states of the transition system.

2. The current state of the ABTA and a transition from the current state of the transition system are labelled with the same action. This is the case of the clauses 8 and 16. In this case, the current transition of the ABTA and the transition from the current state s_i of the transition system to a state $s_{i+1,\,0}$ of the associated semantic transition system are used. The idea is to start the parsing of the formula coded in the semantic transition system.

3. The current state of the ABTA and a transition from the current state of the transition system are labelled with the same action which is preceded by $\neg$ in the ABTA. This is the case of the clauses 9 and 17. In this case, the current transition of the ABTA and the transition from the current state s_i of the transition system to a state $s_{i+1,\,0}$ of the associated semantic transition system are used. The successor node is classified neutral. This allows us to verify the structure of the formula coded in the transition system.

4. The current state of the ABTA and a transition from the current state of the transition system are labelled with different actions where the state of the ABTA is labelled with a negative formula. This is the case of the clauses 10 and 18. In this case, the formula is satisfied, but its structure cannot be verified. Consequently, the current transition of the ABTA and the transition from the current state s_i of the transition system to a next state s_{i+1} are used. This means that, we do not visit the associated semantic transition system.

Finally, the clauses 19, 20, 21, 22, and 23 deal with the case of verifying the structure of the commitment formulae in the sub-transition systems. In these clauses, transitions $s_{i,j} \rightarrow s_{i,j+1}$ are used. We note here that when $s_{i,j}$ has no successor, the formula contained in this state is an atomic formula or a Boolean formula whose all the sub-formulae are atomic (for example $p \wedge q$ where p and q are atomic).

We also need to define the notion of *success* of a run for the correctness of the model checking. To define this notion, we first introduce *positive* and *negative paths*. In an ABTA, every infinite path has a suffix that contains either positive or negative nodes, but not both. Such a path is referred to as *positive* in the former case and *negative* in the latter.

Let $p \in \Phi p$ and let s_i be a state in a transition system T. Then $s_i \text{ ‘ }_T p$ iff $p \in Lab(s_i)$ and $s_i \text{ ‘ }_T \neg p$ iff $p \notin Lab(s_i)$.

Let $s_{i,j}$ be a state in a semantic transition system of a transition system T. Then $s_{i,j} \text{ ‘ }_T p$ iff $p \in Lab'(s_{i,j})$ and $s_{i,j} \text{ ‘ }_T \neg p$ iff $p \notin Lab'(s_{i,j})$.

Definition 5.5 *Let r be a run of ABTA* $B = \langle Q, l, \rightarrow, q_0, F \rangle$ *on a transition system* $T = \langle S, Lab, \wp, L, Act, \rightarrow, s_0 \rangle$. *The run r is successful iff every leaf and every infinite path in r is successful. A successful leaf is defined as follows:*

1- *A positive leaf labeled by* $\langle q, s_i \rangle$ *is successful iff* $s_i \text{ ‘ }_T l(q)$ or $l(q) = \langle \bullet \rangle$ *where* $\bullet \in Act$ *and there is no* s_j *such that* $s_i \rightarrow^{\bullet} s_j$.

2- *A positive leaf labeled by* $\langle q, s_{i,j} \rangle$ *is successful iff* $s_{i,j} \text{ ‘ }_T l(q)$

3- *A negative leaf labeled by* $\langle q, s_i \rangle$ *is successful iff* $s_i \text{ ‘ }_T \neg l(q)$ or $l(q) = \langle \bullet \rangle$ *where* $\bullet \in Act$ *and there is no* s_j *such that* $s_i \rightarrow^{\bullet} s_j$.

4- *A negative leaf labeled by* $\langle q, s_{i,j} \rangle$ *is successful iff* $s_{i,j} \text{ ‘ }_T \neg l(q)$

5- *All neutral leaves are not successful.*

A successful infinite path is defined as follows:

1- *A positive path is successful iff* $\forall f \in F, \exists q \in f$ *such that q occurs infinitely often in the path. This condition is called the Büchi condition.*

2- *A negative path is successful iff* $\exists f \in F, \forall q \in f$, *q does not occur infinitely often in the path. This condition is called the co-Büchi condition.*

We note here that a positive or negative leaf labeled by $\langle q, s \rangle$ such that $l(q) = \langle \bullet \rangle$ where $\bullet \in Act$ and there is no s' such that $s \rightarrow^{\bullet} s'$ is considered a successful leaf because we can not consider it unsuccessful. The reason is that it is possible to find a transition labeled by $\bullet$ and starting from another state s'' in the transition system. If we consider such a leaf unsuccessful, then even if we find a successful infinite path, the run will be considered unsuccessful. However

this is false. We also note that an ABTA B accepts a transition system T iff there is a successful run of B on T.

5.4.2 Translation Procedure

Translating a CTL*^{CA} formula $p = E\phi$ to an ABTA B uses goal-directed rules in to build a tableau for this formula. These rules are conducted in a top-down fashion to determine whether or not states satisfy properties. The tableau is constructed by exhaustively applying the rules contained in Appendix I Figures App I.1, App I.2, App I.3 and App I.4 to p. Then, B can be extracted from this tableau as follows.

First, we generate the states and the transitions. States will correspond to state formulae, with the start state being p. To generate new states from an existing state for a formula p', we determine which rule is applicable to p', starting with $R1$, by comparing the form of p' to the formula appearing in the "goal position" of each rule. Let *rule*(q) denote the rule applied at node q. The labeling function l of states is defined that if q does not have any successor, $l(q) \in \Phi_p$, otherwise, the successors of q are given by *rule*(q). The label of the rule becomes the label of the state q, and the sub-goals of the rule are then added as states related to q by transitions.A tableau for a CTL*^{CA} formula p is a maximal proof tree having p as its root and constructed using rules $R1$-$R27$. If p' results from the application of a rule to p, we say that p' is a child of p in the tableau. The height of a tableau is defined as the length of the longest sequence $<p_0, p_1, \ldots>$, where p_{i+1} is the child of p_i [19].

We give the definition 5.6 for computing the acceptance states F in order to compute the successful run of the generating ABTA.

Definition 5.6 *Let q be a state in an ABTA B and Q the set of all states. Suppose* $\phi = \phi_1 U^+ \phi_2 \in q$[6]. *We define the set F_ϕ as follows*:

$F_\phi = \{q' \in Q \mid (\phi \notin q' \text{ and } X^+\phi \notin q') \text{ or } \phi_2 \in q'\}$.

[6] Here we consider "until" formula because is the formula that allows paths to be infinite.

The acceptance set F is defined as follows:

$F = \{F_\phi \mid \phi = \phi_1 U^+ \phi_2 \text{ and } \exists q \in B, \phi \in q\}$.

According to this definition, a state that contains the formula ϕ or the formula $X^+\phi$ is not an acceptance state. The reason is that according to Definition 5.4, there is a transition from a state containing ϕ to a state containing $X^+\phi$ and vice versa. Therefore, according to Definition 5.5, there is a successful run in the ABTA B. However, we can not decide about the satisfaction of a formula using this run. In an infinite cycle including a state containing ϕ and a state containing $X^+\phi$, we can not be sure that a state containing ϕ_2 is reachable. However, according to the semantics of U^+, the satisfaction of ϕ needs that a state containing ϕ_2 is reachable while passing by states containing ϕ_1.

We show a practical case with propositional formula: $E(G^+F^+p)$ on how a CTL*^{CA} formula is translated to an ABTA. In the context of dialogue game-based agents, this formula says that along some transitions, globally in the future commitment content holds. The first step is to build the tableau for this formula using tableau rules. The first rule we can apply is R27 labeled by "$\vee$" for the "*until*" formula (G^+ is an abbreviation defined from U^+). The second rule is also R27 for F^+p (F^+ is also an abbreviation defined from U^+). Thereafter rules R19 and R24 can be applied. We obtain the tableau illustrated in Table 5.2 where the rule labels are indicated.

Table5.2 The tableau for $E(G^+F^+p)$

$\vee : E(G^+F^+p)$ (1)		
$\vee : E(F^+p, X^+G^+F^+p)$ (2)		
$\Leftrightarrow : E(p, X^+G^+F^+p)$ (3)		$X^+ : E(X^+F^+p, X^+G^+F^+p)$ (4)
p (5)	$X^+ : E(X^+G^+F^+p)$ (6)	$\vee : E(F^+p, G^+F^+p)$ (7)
	$E(G^+F^+p)$	$E(F^+p, X^+G^+F^+p)$

The ABTA obtained from this tableau is illustrated in Figure 5.2. In this ABTA, states (1), (3), (5) and (6) are the acceptance states according to Definition 5.6. The formula ϕ we consider is the following: ϕ = True U^+ p $\equiv F^+p$. Notice that ϕ and $X^+\phi$ do not appear in these states. State (5) is the acceptance state in the finite case. On the other hand, ϕ appears in states (2) and (7), and $X^+\phi$ appears in state (4). Therefore, these states are not in F_ϕ. The path Π = (1, (2, 4, 7)*) is not a valid proof of $E(G^+F^+p)$. However, a path that visits infinitely often the states (1), (3) and (6) is a valid (infinite) proof. The reason is that in such a path there is always a chance to meet the proposition p (state (3)). Therefore, this path satisfies the Büchi condition. The Büchi condition is not satisfied in the path Π since there is no chance to visit infinitely often a state containing p.

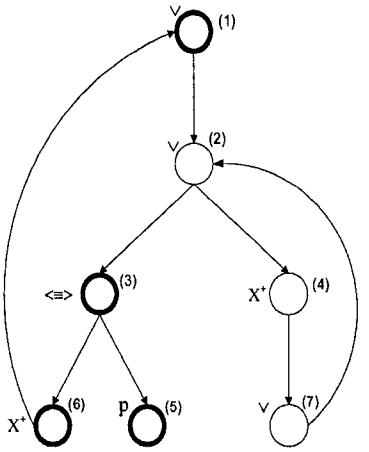

Figure 5.2 The ABTA of formula $E(G^+F^+p)$

5.4.3 Model Checking Algorithm

The idea behind our model checking algorithm is to explore the product graph of an ABTA for CLT*^{CA} and a transition system for a dialogue game. This algorithm is *on-the-fly* (or *local*) algorithm consisting of checking if a transition system is accepted by an ABTA. This model checking is reduced to the emptiness of the Büchi automata [46].

Let $T = < S, Lab, \wp, L, Act, \rightarrow, s_0 >$ be a transition system for a dialogue game and let $B = <Q, l, \rightarrow, q_0, F>$ be an ABTA for CTL*^{CA}. The procedure consists of building the ABTA product $B_\otimes$ of T and B while checking if there is a successful run in $B_\otimes$. The existence of such a run means that the language of $B_\otimes$ is non-empty. The automaton $B_\otimes$ is defined as follows: $B_\otimes = <Q \times S, \rightarrow_{B\otimes}, q_{0B\otimes}, F_{B\otimes}>$. There is a transition between two nodes $<q, s>$ and $<q', s'>$ iff there is a transition between these two nodes in some run of B on T. Intuitively; $B_\otimes$ simulates all the runs of the ABTA. The set of accepting states $F_{B\otimes}$ is defined as $q_{0B\otimes} \in F_{B\otimes}$ iff $q \in F$.

Unlike the algorithms proposed in [12], our algorithm uses only one depth-first search (DFS) instead of two. This is due to the fact that our algorithm explores directly the product graph using the sign of the nodes (positive, negative or neutral). Another difference is that our algorithm does not distinguish between recursive and non-recursive nodes. Therefore, we do not take into account the strongly-connected components in the ABTA, but we use a marking algorithm that works on the product graph.

The pseudo-code of this algorithm is given in table 5.3. The idea is to construct the product graph while exploring it. However, in order to make it easy to understand, we omit the instructions relative to the addition of nodes in the product graph. The construction procedure is directly obtained from Definition 5.4. The algorithm uses the label of nodes in the ABTA, and the transitions in the product graph obtained from the transition system and the ABTA as explained in Definition 5.4.

In order to decide if the ABTA contains an infinite successful run, all the explored nodes are marked "visited". Thus, when the algorithm explores a visited node, it returns false if the infinite path is not successful. If the node is not already visited, the algorithm tests if it is a leaf. In this case, it returns false if the node is a non-successful leaf. If the explored node is not a leaf, the algorithm calls recursively the function DFS in order to explore the successors of this node. If this node is labeled by "$\wedge$", and signed neutrally or positively, then DFS returns false if one of the

successors is false. However, if the node is signed negatively, DFS returns false if all the successors are false. A dual treatment is applied when the node is labeled by "$\vee$". We note that if the DFS does not explore a false node (i.e. it does not return false), then it returns true.

Theorem 5.1 (algorithm's correctness) *Let B an ABTA and T a transition system. DFS(q_0, s_0) returns true if and only if T is accepted by B.*

Theorem 5.2 (technique's soundness and completeness) *Let ψ be a CTL*^{CA} formula and B_ψ the ABTA obtained by the translation procedure, and let T be a transition system that represents a dialogue game protocol. Then s_0 ‘ ψ iff T is accepted by B_ψ.*
The proofs of these theorems are developed in [4].

5.5 Termination

Since the translation procedure is based on tableau rules, we need to prove the finiteness of the tableau. The methodology we follow is inspired by [18].

If σ_2 is a CTL*^{CA} formula resulting from the application of a rule to a CTL*^{CA} formula σ_1, then we say that σ_2 is a child of σ_1 in the tableau and σ_1 is the parent of σ_2. The *height* of a tableau [18] is defined as the length of the longest sequence $<\sigma_0, \sigma_1, \ldots>$, where σ_i is the parent of σ_{i+1}. To prove the finiteness of a tableau, we will establish that each formula has a maximum height tableau.

Intuitively, to show the finiteness of the tableau, we will define a strict ordering relation $\prec$ between CTL*^{CA} formulae and then show that: **1)** if σ_1 is the parent of σ_2, then $\sigma_1 \prec \sigma_2$; **2)** the strict ordering relation $\prec$ has no infinite ascending chains.

Table 5.3 Model checking algorithm

```
DFS(v = (q, s)): boolean {
    if v marked visited {
      if (sign(v) = "+" and not accepting(v)) or (sign(v) = "-" and accepting(v))
        return false
    } // end of if v marked visited
    else {
      mark v visited
      switch(l(q)) {
          case (p ∈ Φp):
              switch(sign(v)) {
                  case("+"): if s is a sub-state and l(q) ∉L'(s) return false
                  case("-"): if s is a sub-state and ¬l(q)) ∉L'(s) return false
                  case("neutral"): return false
              } // end of switch(sign(v))
          case(∧):
              if s is a leaf return false
              else
                  switch(sign(v)) {
                      case(neutral): for all v'' ∈ {v' / v →B⊗ v'}
                                     if not DFS(v'') return false
                      case("+"): for all v'' ∈ {v' / v →B⊗ v'}
                                 if not DFS(v'') return false
                      case("-"): for all v'' ∈ {v' / v →B⊗ v'}
                                 if DFS(v'') return true else return false
                  } // end of switch(sign (v))
          case(∨):
              if s is a leaf return false
              else
                  switch(sign(v)) {
                      case(neutral): for all v'' ∈ {v' / v →B⊗ v'}
                                     if DFS(v'') return true else return false
                      case("+"): for all v'' ∈ {v' / v →B⊗ v'}
                                 if DFS(v'') return true else return false
                      case("-"): for all v'' ∈ {v' / v →B⊗ v'}
                                 if not DFS(v'') return false
                  } // end of switch(sign (v))
          case(<•>):
              if s is a leaf return true
              else for the v'' ∈ {v' / v →B⊗ v'} if not DFS(v'') return false
          case(X+, PCAg, ACAg, <≡>, ?):
              if s is a leaf return false
              else for the v'' ∈ {v' / v →B⊗ v'} if not DFS(v'') return false
      } // end of switch(l(q))
    } // end of else
    return true }
```

The ordering relation $\prec$ should reflect the fact that applying tableau rules results in shorter formulae or recursive formulae. The idea is to prove that the number of nodes of the ABTA is finite. Therefore, the definition of this ordering is based either on the fact that formulae are recursive or on the length of formulae. We notice that in the case of recursive formulae, we obtain cycles which are infinite paths on a finite number of nodes. The length of a formula is defined inductively as follows:

Definition 5.7 *The length of a formula ψ denoted by $|\psi|$ is the number of variables and operators in ψ i.e.:*

$|\psi| = 1$ if ψ *is an atomic formula*

$|\neg\psi| = 1 + |\psi|$

$|\psi_1 \wedge \psi_2| = 1 + |\psi_1| + |\psi_2|$

$|\psi_1 \vee \psi_2| = 1 + |\psi_1| + |\psi_2|$

$|?\psi| = 1 + |\psi|$

$|\psi_1 \therefore \psi_2| = 1 + |\psi_1| + |X^+(\neg\psi_1 \vee \psi_2)|$

$|X\psi| = 1 + |\psi|$ *where* $X \in \{X^+, X^-\}$

$|\psi_1 \, U \, \psi_2| = 1 + |\psi_1| + |\psi_2|$ *where* $(U, X) \in \{(U^+, X^+), (U^-, X^-)\}$

$|PC(Ag_1, Ag_2, t, \psi)| = 1 + |\psi|$

$|C(Ag_1, SC(Ag_1, Ag_2, t, \psi))| = 1 + |PC(Ag_1, Ag_2, t, \psi)|$

$|Challenge(Ag_2, PC(Ag_1, Ag_2, t, \psi))| = 1 + |PC(Ag_2, Ag_1, t', ?\psi)|$

$|Accept(Ag_2, SC(Ag_1, Ag_2, t, \psi))| = 1 + |PC(Ag_2, Ag_1, t', \psi)|$

$|Refuse(Ag_2, SC(Ag_1, Ag_2, t, \psi))| = 1 + |PC(Ag_2, Ag_1, t', \neg\psi)|$

$|Justify(Ag_1, PC(Ag_1, Ag_2, t, \psi' \therefore \psi))| = 1 + |PC(Ag_1, Ag_2, t', \psi' \therefore \psi)|$

$|Attack(Ag_2, PC(Ag_1, Ag_2, t, \psi' \therefore \psi))| = 1 + |PC(Ag_2, Ag_1, t', \psi' \therefore \neg\psi)|$

The ordering relation $\prec$ is defined as follows:

Definition 5.8 *Let $\sigma_1 = E(\psi_1)$ and $\sigma_2 = E(\psi_2)$ be two CTL*^{CA} formulae. Then, $\sigma_1 \prec \sigma_2$ holds if*

1- σ_1 , σ_2

2- σ_1 " σ_2 *and* $|\psi_1| > |\psi_2|$.

where σ_1 , σ_2 *iff* $X\psi_1$ *appears in* ψ_2

The first clause is used when we have a recursive formula (this means that an *"until"* formula). "$\prec$" is irreflexive, asymmetric and transitive. The proof is straightforward from the definition since > and , are strict ordering relations.

In what follows, the notation $\sigma_1 \rightarrow_R \sigma_2$ means that σ_1 is the parent of σ_2 using a tableau rule R. We have the several lemmas. The proof of these lemmas can be found in the appendix II.

Lemma 5.1 *Let* $\sigma_1 = \mathrm{E}(\psi_1)$ *and* $\sigma_2 = \mathrm{E}(\psi_2)$ *be two DCTL*$_{CAN}$ formulae. Then:*

$\sigma_1 \rightarrow_R \sigma_2 \Rightarrow \sigma_1 \prec \sigma_2$.

To show that the ordering relation has no infinite ascending chains, we use the notion of Fischer-Ladner closure of a formula ψ ($CL(\psi)$) [21]. The idea underlying the definition of this notion is to prove that if a tableau has a root ψ, then all formulae ψ' of this tableau have a formula in $CL(\psi)$ (i.e. $\psi' \in CL(\psi)$). Furthermore, if we prove that $CL(\psi)$ is a finite set, then we conclude that each formula appearing in a given tableau belongs to a finite set. This result will be very helpful to prove that the ordering relation $\prec$ has no infinite ascending chains.

Definition 5.9 *Let* ψ *be a CTL*^{CA} formula. The Fischer-Ladner closure of* ψ, $CL(\psi)$ *is the smallest set such that the following hold:*

If ψ *is an atomic formula then* $\{\psi\} \subseteq CL(\psi)$

If $\psi = \neg\psi_1$ *then* $CL(\psi_1) \subseteq CL(\psi)$ and $\{\neg\psi_1\} \subseteq CL(\psi)$

If $\psi = \psi_1 \wedge \psi_2$ *then* $CL(\psi_1) \subseteq CL(\psi)$ *and* $CL(\psi_2) \subseteq CL(\psi)$ *and* $\{\psi_1 \wedge \psi_2\} \subseteq CL(\psi)$

If $\psi = \psi_1 \vee \psi_2$ *then* $CL(\psi_1) \subseteq CL(\psi)$ and $CL(\psi_2) \subseteq CL(\psi)$ *and* $\{\psi_1 \vee \psi_2\} \subseteq CL(\psi)$

If $\psi = ?\psi_1$ *then* $CL(\psi_1) \subseteq CL(\psi)$ *and* $\{?\psi_1\} \subseteq CL(\psi)$

If $\psi = \psi_1 \therefore \psi_2$ *then*

$CL(\psi_1) \subseteq CL(\psi)$ *and* $CL(X^+(\neg\psi_1 \vee \psi_2)) \subseteq CL(\psi)$ *and* $\{\psi_1 \therefore \psi_2\} \subseteq CL(\psi)$

If $\psi = X\psi_1$ *then* $CL(\psi_1) \subseteq CL(\psi)$ *and* $\{X\psi_1\} \subseteq CL(\psi)$ *where* $X \in \{X^+, X^-\}$

If $\psi = \psi_1 U \psi_2$ *then*

$CL(\psi_1) \subseteq CL(\psi)$ *and* $CL(\psi_2) \subseteq CL(\psi)$ *and* $CL(\mathrm{X}(\psi_1 U \psi_2)) \subseteq CL(\psi)$

and $\{\psi_1 \mathrm{U} \psi_2\} \subseteq CL(\psi)$ *where* $(\mathrm{U}, \mathrm{X}) \in \{(U^+, X^+), (U^-, X^-)\}$

If $\psi = PC(Ag_1, Ag_2, t, \psi_1)$ *then*

$CL(\psi_1) \subseteq CL(\psi)$ *and* $\{PC(Ag_1, Ag_2, t, \psi_1)\} \subseteq CL(\psi)$

If $\psi = C(Ag_1, PC(Ag_1, Ag_2, t, \psi_1))$ *then*

$CL(PC(Ag_1, Ag_2, t, \psi_1)) \subseteq CL(\psi)$ *and* $\{C(Ag_1, PC(Ag_1, Ag_2, t, \psi_1))\} \subseteq CL(\psi)$

If $\psi = Challenge(Ag_1, PC(Ag_1, Ag_2, t, \psi_1))$ *then*

$CL(PC(Ag_1, Ag_2, t', ?\psi_1)) \subseteq CL(\psi)$

and $\{Challenge(Ag_1, PC(Ag_1, Ag_2, t, \psi_1))\} \subseteq CL(\psi)$

If $\psi = Accept(Ag_1, PC(Ag_1, Ag_2, t, \psi_1))$ *then*

$CL(PC(Ag_1, Ag_2, t', \psi_1)) \subseteq CL(\psi)$ s

and $\{Accept(Ag_1, PC(Ag_1, Ag_2, t, \psi_1))\} \subseteq CL(\psi)$

If $\psi = Refuse(Ag_1, PC(Ag_1, Ag_2, t, \psi_1))$ *then*

$CL(PC(Ag_1, Ag_2, t', \neg\psi_1)) \subseteq CL(\psi)$

and $\{Refuse(Ag_1, PC(Ag_1, Ag_2, t, \psi_1))\} \subseteq CL(\psi)$

If $\psi = Justify(Ag_1, PC(Ag_1, Ag_2, t, \psi_2 \therefore \psi_1))$ *then*

$CL(PC(Ag_1, Ag_2, t', \psi_2 \therefore \psi_1)) \subseteq CL(\psi)$

and $\{Justify(Ag_1, PC(Ag_1, Ag_2, t, \psi_2 \therefore \psi_1))\} \subseteq CL(\psi)$

If $\psi = Attack(Ag_2, PC(Ag_1, Ag_2, t, \psi_2 \therefore \neg\psi_1))$ *then*

$CL(PC(Ag_2, Ag_1, t', \psi_2 \therefore \neg\psi_1)) \subseteq CL(\psi)$

and $\{Attack(Ag_2, PC(Ag_1, Ag_2, t, \psi_2 \therefore \neg\psi_1))\} \subseteq CL(\psi)$

Lemma 5.2 *Let* ψ be a formula, then $CL(\psi)$ is finite and bounded in size by $2|\psi|$.

The next lemma establishes the link between tableau rules and Fischer-Ladner closure of formulae.

Lemma 5.3 *Let* $\sigma_1 = E(\Phi, \psi_1)$ *and* $\sigma_2 = E(\Phi, \psi_2)$ *be two* CTL^{*CA} *formulae. Then:*

$\sigma_1 \rightarrow_R \sigma_2 \Rightarrow CL(\psi_2) \subseteq CL(\psi_1)$.

Intuitively, $\sigma_i \prec \sigma_j$ holds if σ_i is an ancestor of σ_j in some tableau, i.e. if there are rules *Ri*, ..., *Rj* such that: $\sigma_i \rightarrow_{Ri} \sigma_{i+1} \ldots \rightarrow_{Rj} \sigma_j$. We have the following lemma

Lemma 5.4 *The ordering relation* $\prec$ *has no infinite ascending chains.*

Theorem 5.3 *For any* CTL^{*CA} *formula* σ_1, *there is a maximum height tableau has* σ_1 *as a root.*

We now discuss the worst-case time complexity of our model checking.

Lemma 5.5 *Let* ψ *be a* CTL^{*CA} *formula, and let* $B_\psi = <Q, l, \rightarrow, q_0, F>$ be *the ABTA obtained by the translation procedure. Then* $|B_\psi| < 2^{|\psi|}$.

5.6 Case Study

5.6.1 Protocol Description and Verification for Communities of WS

We use our model checking technique to verify our communities' protocol: *PNP-CWS*. *PNP-CWS* is a dialogue game-based protocol allowing Web services implemented as argumentative agents to interact with each other in a negotiation setting. Agents can negotiate their participation in composite Web services and persuade each other to perform some actions. *PNP-CWS* is specified using two special moves: *refusal* and *acceptance* as well as five dialogue games: *entry game* (to open the interaction), *offer game*, *challenge game*, *justification game*, and *attack game*. In order to clarify the *PNP-CWS* for our model checking, we redefine *PNP-CWS* with a BNF-like grammar where "|" is the choice symbol and ";" the sequence symbol as follows:

PNP-CWS =*entry game*; WSDG
WSDG = refusal move|acceptance move | Ch | Att
Ch = *challenge game*;*justification game*; (WSDG | refusal move)
Att = *attack game*; (WSDG | refusal move)

Each game is specified by a set of moves using a set of logical rules. The rules were described in Chapter 4. We use a graphical representation, which helps to understand the protocol dynamics. Figure 5.3 illustrates this representation as a finite state machine.

Many properties can be checked in this protocol, such as deadlock freedom (a safety property), and aliveness (something good will eventually happen). Deadlock freedom can be expressed in our CTL^{*CA} logic as follows:

$$AG^{+}(Act(Ag_i, PC(Ag_1, Ag_2, \mathrm{P}))$$

which states that there is always a possibility for an action. An example of aliveness can be expressed by the following formula:

$$EF^{+}(Accept(Ag_2, PC(Ag_1, Ag_2, \mathrm{P})) \vee Refuse(Ag_2, PC(Ag_1, Ag_2, \mathrm{P})))$$

which states that there is a possibility to achieve some good states (accept or refuse). Another example of aliveness property is given by the following formula stating that if there is a challenge, a justification will eventually follow:

$$AG^{+}(\text{Challenge}(Ag_2, PC(Ag_1, Ag_2, t, \phi)) \Rightarrow F^{+}Justify(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi)))$$

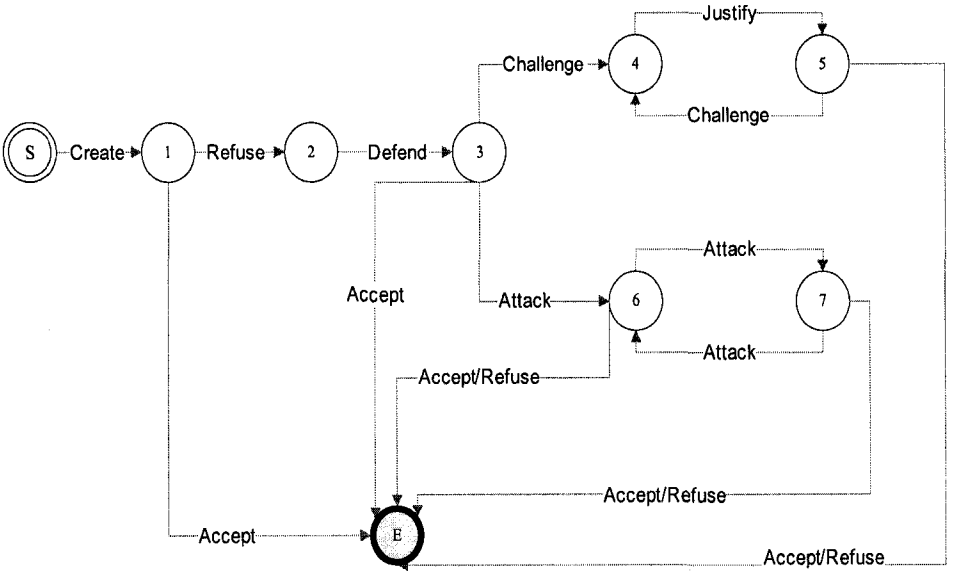

Figure 5.3 The *PNP-CWS* Protocol

The two first properties are relatively easy to check. We focus in this section on the third property. In order to simplify the formula, we use *Ch* for *Challenge* and *Jus* for *Justify*. The first step in our technique is the transformation of the formula to a tableau. The tableau of this formula is illustrated by Table 5.4. The second step is building the associated ABTA. The ABTA of this formula is given by Figure 5.4. This formula is equivalent to:

$$AG^{+}(\neg Ch(Ag_2, PC(Ag_1, Ag_2, t, \phi)) \vee F^{+}Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi)))$$

To build the tableau, the first rule we can apply is *R*6 labeled by "¬". We obtain then the formula (2) of Table 5.4. From this formula we obtain the formula Φ that we consider in order to compute the acceptance states:

$$\Phi = F^{+}(Ch(Ag_2, PC(Ag_1, Ag_2, t, \phi)) \wedge G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))))$$

In the ABTA of Figure 5.4 state (1) and states from (3) to (18) are the acceptance states according to Definition 5.6. States (2) and (4) are not acceptance states. Because only the first state is labeled by ¬, all finite and infinite paths are negative paths. Consequently, the only infinite path that is a valid proof of the formula Φ is (1, (2, 4)*). In this path there is no acceptance state that occurs infinitely often. Therefore, this path satisfies the Büchi condition. The path visiting the state (3) and infinitely often the state (9) does not satisfy the formula because there is a challenge action (state (3)), and globally no justification action of the content of the challenged propositional commitment (state (9)).

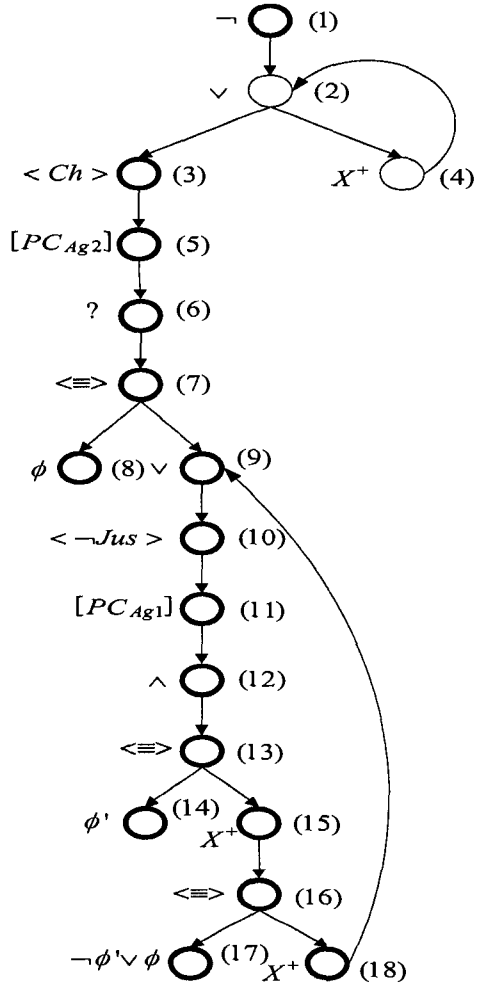

Figure 5.4 The ABTA for case study formula

$AG^{+}(Ch(Ag_2, PC(Ag_1, Ag_2, t, \phi)) \Rightarrow F^{+}Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi)))$

Table 5.4 The tableau for

$$AG^{+}(Ch(Ag_2, PC(Ag_1, Ag_2, t, \phi)) \Rightarrow F^{+}Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi)))$$

$\neg: AG^{+}(\neg Ch(Ag_2, PC(Ag_1, Ag_2, t, \phi)) \vee F^{+}Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi)))$ (1)

$\vee: EF^{+}(Ch(Ag_2, PC(Ag_1, Ag_2, t, \phi)) \wedge G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi'))))$ (2)

$< Ch >: E(Ch(Ag_2, PC(Ag_1, Ag_2, t, \phi)) \wedge G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))))$ (3)

$< X^{+} >: EX^{+}(F^{+}(Ch(Ag_2, PC(Ag_1, Ag_2, t, \phi)) \wedge G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi)))))$ (4)

$[PC_{Ag2}]: E(PC(Ag_2, Ag_1, t, ?\phi) \wedge G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi'))))$ (5)

$EF^{+}(Ch(Ag_2, PC(Ag_1, Ag_2, t, \phi)) \wedge G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))))$ (2)

$?: E((?\phi) \wedge G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))))$ (6)

$<\equiv>: E(\phi \wedge G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))))$ (7)

φ (8) $\vee: E(G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))))$ (9)

$< \neg Jus >: E(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi)), X^{+}G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))))$ (10)

$[PC_{Ag1}]: E(PC(Ag_1, Ag_2, t, \phi' \therefore \phi), X^{+}G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))))$ (11)

$\wedge: E(\phi' \therefore \phi, X^{+}G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))))$ (12)

$<\equiv>: E(\phi', X^{+}(\neg\phi' \vee \phi), X^{+}G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))))$ (13)

ϕ' (14) $X^{+}: E(X^{+}(\neg\phi' \vee \phi), X^{+}G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))))$ (15)

$<\equiv>: E((\neg\phi' \vee \phi), X^{+}G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))))$ (16)

$\neg\varphi' \vee \varphi$ (17) $X^{+}: E(X^{+}G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))))$ (18)

$E(G^{+}(\neg Jus(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi))))$ (9)

Figure 5.5 illustrates the automaton $B_{\otimes}$ resulting from the product of the transition system of Figure 9 to which we add the internal states to describe the syntax exactly as illustrated in Figure 5.1 and the ABTA of Figure 5.4. We will use TS [12] to denote the protocol and ABTA [13] to denote the ABTA of Figure 5.4. In order to check if the language of this automaton is empty, we check if there is a successful run. The idea is to verify if $B_{\otimes}$ contains an infinite path visiting the state (3) and infinitely often the state (9) of ABTA [13]. If such a path exists, then we conclude that the formula is not satisfied by TS [12]. Indeed, the only infinite path of $B_{\otimes}$ is

successful because it does not touch any accepted state and all leaves are also successful. For instance, the leaf labeled by ($<Ch>,s_0$) is successful since there is no state s_i such that $s_0 \rightarrow^{Ch} s_i$. The leaf labeled by ($\neg\phi' \vee \phi$, $s_{3,4}$) is successful because it is a positive leaf and $s_{3,4}$ ' $\neg\phi' \vee \phi$. Therefore, TS [12] is accepted by ABTA [13]. Consequently, TS [12] satisfies the formula and respects the structure of challenge and justification actions.

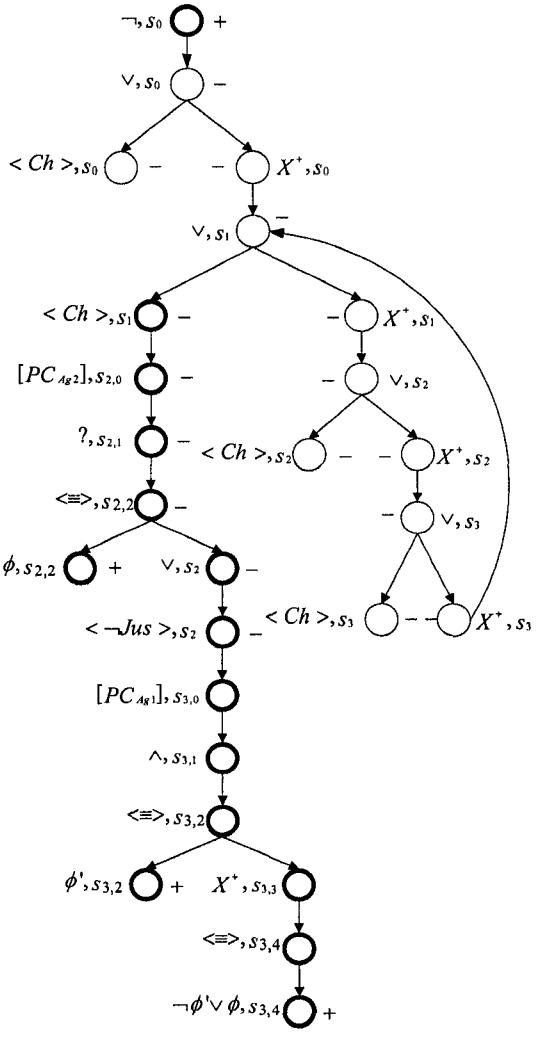

Figure 5.5 The ABTA product graph

5.6.2 Implementation

We implemented this case study using a modified and enhanced version of the Concurrency Workbench of New Century (CWB-NC). This model checker supports GCTL*, which is close to our logic (without propositional commitments) and allows modeling concurrent systems using process algebra Calculus of Communicating Systems (CCS) developed by Milner [32]. CCS language is a paradigmatic process algebras language, which is a prototype specification language for reactive systems. For this reason, CCS language can be used not only to describe implementations of processes, but also specifications of their expected behaviors.

To use CCS as the design language to descript *PNCWS* protocol, we need first to describe CCS formally for this protocol. Let A be the set of actions performed on propositional commitments we consider in our logic. With every $a \in A$ we associate a complementary action $'a$. Intuitively, an action a will represent the receipt of an input action, while $'a$ will represent the deposit of an output action. The syntax of CCS for *PNCWS* protocol is given by the following BNF grammar:

$$P ::= nil \mid \alpha(\phi).P \mid (P + P) \mid (P \mid P) \mid proc\ C = P$$

"." Represents the prefixing operator, "+" is the choice operator, "|" is the parallel operator and " $proc =$ " is used for defining processes. The semantics can be defined using operational semantics in the usual way. $\alpha(\phi).P$ is the processes of performing the action α on the propositional commitment content ϕ and then evolves into process P. For simplification reasons, we consider only the commitment content and we omit the other arguments. In addition, we abstract away from the internal states used to check the content syntax, and we focus only on verifying the properties from a semantic perspective. $P+Q$ is the process which non-deterministically makes the choice of evolving into either P or Q. $P \mid Q$ is the process which evolves in parallel into P and Q.

To verify *PNCWS* we need to model the protocol and the agents using this protocol. For this reason, we need to define two processes: the *states process* describing the protocol dynamics and the *agent process* describing the agent legal decisions. These two processes are as follows:

The definition of the states process:

```
proc Spec = create(ϕ).S1
proc Accept = accept(ϕ).Spec
proc Accept' = 'accept(ϕ).Spec
proc Refuse = refuse(ϕ).Spec
proc Refuse' = 'refuse(ϕ).Spec
proc S1 = 'refuse(ϕ).S2 + Accept'
proc S2 = defend(ϕ').S3
proc S3 = 'challenge(ϕ').S4 + 'attack(ϕ).S6 + 'accept(ϕ').Spec
proc S4 = justify(ϕ).S5
proc S5 = 'challenge(ϕ).S4 + 'Accept +'Refuse
proc S6 = attack(ϕ').S7 + Accept +Refuse
proc S7 = 'attack(ϕ).S6 + 'accept(ϕ').Spec + 'refuse(ϕ').Spec
set Internals = {create, challenge, justify, accept, refuse, attack, defend}
```

The definition of the agent process:

```
proc Ag = 'create(ϕ).Ag +
      create(ϕ).('refuse(ϕ).Ag + 'accept(ϕ).Ag) +
      refuse(ϕ).(Ag + 'defend(ϕ').Ag) +
      defend(ϕ').('challenge(ϕ').Ag + 'attack(ϕ).Ag + 'accept(ϕ').Ag) +
      challenge(ϕ).'justify(ϕ').Ag +
      justify(ϕ').('challenge(ϕ').Ag + 'accept(ϕ').Ag +'refuse(ϕ').Ag) +
      attack(ϕ').('attack(ϕ).Ag + 'accept(ϕ').Ag +'refuse(ϕ').Ag) +
      accept(ϕ).Ag
```

Figure 5.6 illustrates the result of checking the liveness property discussed in the previous section stating that if there is a challenge, a justification will eventually follow. The figure

illustrates also the result of checking deadlock property. The other properties (reachability of accept and refuse) are also successfully checked.

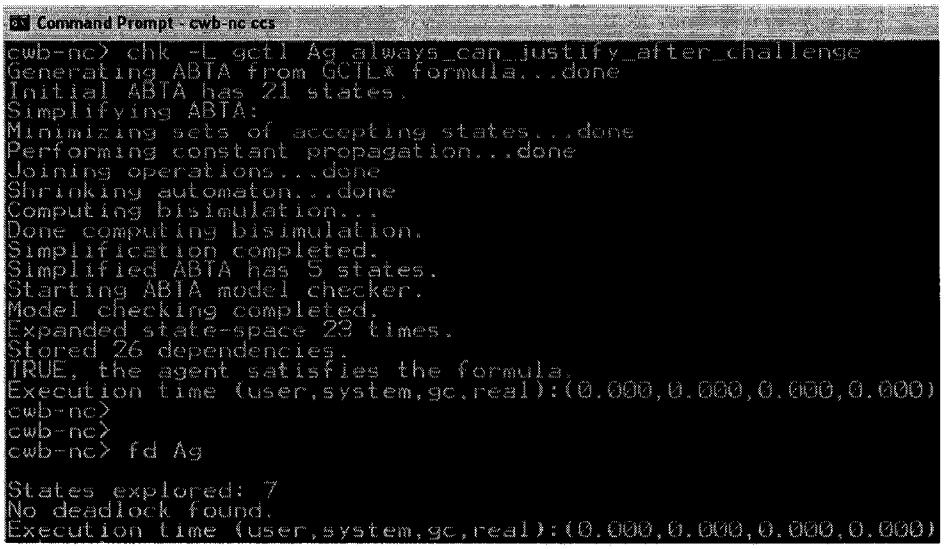

Figure 5.6 Result of Checking Liveness and Deadlock Properties

5.7 Conclusion

In this chapter, we have addressed the verification problem of argumentative agent-based community of Web services, in which knowledge-driven agents communicate by reasoning about dialogue game protocols. We proposed a new model checking technique allowing for the verification of both the correctness of the protocols and the agents' compliance to the structure of the communicative acts. This technique uses a combination of an automata-based and a tableau-driven algorithm to verify temporal and action specification. The formal properties to be verified are expressed in CTL^{*CA} logic and translated into ABTA using tableau rules. We provided correctness and completeness results and we proved that this model checking algorithm working on a product graph is an efficient on-the-fly procedure that always terminates. We applied the proposed technique to a case study describing the verification of some properties in an agent-based community of Web services.

Chapter 6. Conclusion

6.1 Contributions

The objective of this thesis is the specification of argumentative agent-based communities of Web services and the development of techniques for formal verification of communication protocols of agent-based Web services using model checking. The main contributions of this work are summarized below:

- Theoretical contributions:
  1. Formal specification of communities of Web services along with dialogue game-based protocols for communicating services.
  2. Definition of a logic extending CTL* for communication protocols along with its model checking algorithm for the verification of interaction protocols for the communities of Web services.
- Practical contributions:
  1. Implementation of the specified communities and communication protocols using Jadex BDI reasoning engine.
  2. Simulation of the verification model using a modified and enhanced version of CWB-NC model checker.

6.2 Discussions

In our architecture, we have incorporated a new dialogue game approach and protocols for agent communication and interactive Web services. Agents that represent Web services within communities have autonomous, decision-making, and social abilities. They react to the

environmental changes, but they have also pro-activity characteristics allowing them to take initiatives. In this setting, communication is an important issue to be considered. FIPA-ACL is one of the widely used approaches for agent communication engineering. It proposed a set of agent communication protocols to exchange or request information. Although FIPA-ACL has been given well-defined semantics, it is not flexible enough for agents who discuss complicated negotiations with other agents for conflicts of interest. Like Contract Net Protocol (CNP), one of the FIPA-ACL communication protocols, FIPA-ACL protocols define a sequence of steps that agents can follow to exchange information. Contract Net Protocol (SC00029H[7]) is used for bidding between one initiator and several participant agents. There are four steps in a sequence of CNP: 1. Initiator sends out a call for proposals (CFP). 2. Each participant reviews CFP's and bids on feasible ones. 3. Initiator chooses the best bid and awards a contract to that participant. 4. Initiator rejects other bids. This sequence is a simple negotiation procedure.

However, for more complicated conversations, such as one agent questions another agent's proposal, CNP could not support it. Dialogue games were introduced to increase agents' communication abilities and to make the agents negotiate complex contents. Compared to FIPA-ACL protocols, dialogue game protocols have more choices and more flexibility. For example, in our framework for communities of Web services, after entry game, agents who are invited to join a community can reply with acceptance or refusal. If acceptance is received by the initial agent, the negotiation procedure is started. Otherwise, the conversation is ended. For negotiation procedure, after an agent has received an offer, it accepts the offer or refuses it. An agent also can challenge it or attack it based on its knowledge. This gives agents more chances to make a good deal. This negotiation procedure takes place autonomously based on agent's beliefs and goals.

Integrating Web services and agent technologies seems to be promising. The framework presented in this thesis, which uses argumentative agents to argue, negotiate, and reason about Web services, helps Web services in being better organized within communities and in achieving

[7] http://www.fipa.org/specs/fipa00029/SC00029H.html

the goal for which they are conceived. Agents in communities are autonomous, proactive, reactive, and intelligent and perform their duties without human intervention. They use argumentative techniques to reach a decision and to inform, convince, and negotiate with peers. They act as Web services representatives', reason on their behalf and seek scenarios which maximize their profit. Metadata on contents and features of the Web services are presented within the state of the argumentative agent to which the Web services are associated. These agents are equipped with beliefs, including functionality, security, and QoS information, as well as the argumentation capabilities to persuade a Web service to join a community, and to negotiate its participation in a given composition.

For the verification issues, most agent-based model checking systems are based only on temporal logics. This thesis proposes an approach called tableau-based model checking to verify dialogue game protocols using both temporal and dynamic logic. Our model checking procedure can verify whether or not the dialogue game satisfies a given property. Also, it can verify if the tableau semantics of communicative acts is respected. The idea is to integrate the semantics in the specification of the protocols, and then to propose a parsing method to verify that protocol specification respects the semantic definition. The approach checks the agents' compliance with the semantics without taking into account the agents' specifications. In one word, our procedure verifies not only the correctness of the protocol relative to the properties, but also the conformance of agents to the semantics of the communicative acts.

6.3 Future Work

This thesis is about specifying and verifying communities of Web services using argumentative agents. It opens the door for several research opportunities. Future research may include extending our method for services' interaction to exploit business models, which specify additional relationships between services. Another opportunity is to build a Web-agent bridge

framework, extending communities' management scope by creating more sophisticated negotiation strategies, and developing intelligent model checking tools. In addition, the following points could be considered in the future:

- Our implementation is under Jadex BDI agent platform. We use agents' Directory Facilitator (DF) instead of real UDDIs to search for Web services register functions. Using UDDI will provide more accurate information. Also, we specified Web services as agents, but developing a complete framework for Web-based agent applications is still needed.
- Human beings' activities are more and more involved into Web services. However, our system does not have functions to communicate with humans and to manage users interests and preferences. In the future, we plan to extend our concept of communities and build a new platform for Web services: not only to attract Web services but also to try humans' contributions.
- Our system only provides one negotiation method. Several negotiation strategies for different interests could be added in the future for more sophisticated interactions. Furthermore, semantics of the dialogue games for negotiation strategies should be developed.
- In many circumstances, counter-examples, showing a non-satisfied formula in a model, are very useful for understanding why a formula is false. Unfortunately, we cannot generate counter-examples automatically. Thus, the effective generation of human-readable counter-examples needs to be developed.
- Although our research is not about designing the model checking tools, during the study, we found that model checkers for MAS are not very flexible. These model checkers only can check the formulae using one specific logic. Developing model checking tools for MAS with intelligent learning abilities are still needed.

References

1. Bellifemine, F., G. Caire, and D. Greenwood. 2007. *Developing multi-agent systems with JADE.* West Sussex, England: John Wiley & Sons, Ltd.

2. Benatallah, B., F. Casati, and F. Toumani. 2006. *Representing, analysing and managing Web service protocols.* Data and Knowledge Engineering 58 (3) p: 327-57.

3. Bentahar, J., J.-J. Meyer, and W. Wan. 2008. *Model Checking Communicative Agent-based Systems.* (Accepted in Knowledge-based Systems, Elsevier)

4. Bentahar, J. 2005. *A pragmatic and semantic unified framework for agent communication.* PhD thesis, Laval University, Canada.

5. Bentahar, J., J. Labban, and B. Moulin. 2007. *An argumentation-driven model for efficient and secure negotiation.* Proceedings of The International Conference on Group Decision and Negotiation.

6. Bentahar, J., Z. Maamar, W. Wan, D. Benslimane, P. Thiran, and S. Subramanian. 2008. *Argumentative agents for communities of Web services.*

7. Bentahar, J. Moulin,B., and J. -J C. Meyer. 2006. *A New Model Checking Approach for Verifying Agent Communication Protocols.* Electrical and Computer Engineering, 2006. CCECE '06. Canadian Conference on May 2006 p. 1586 - 1590

8. Bentahar, J., Z. Maamar, D. Benslimane, and P. Thiran. 2007. An argumentation framework for communities of Web services. *IEEE Intelligent Systems* 22, (6) p. 75-83.

9. Bentahar, J., B. Moulin, J. -J C. Meyer, and B. Chaib-draa. 2005. *A computational model for conversation policies for agent communication.* Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2 (AAMAS'04) p. 792-799

10. Besnard, P., and A. Hunter. 2001. *A logic-based theory of deductive arguments.* Artificial Intelligence 128, (1-2) p. 203-35.

11. Bhat, G., and R. Cleaveland. 1996. *Efficient model checking via the equational & μ-calculus.* Paper presented at Proceedings 11th Annual IEEE Symposium on Logic in Computer Science, p. 304-312

12. Bhat, S., R. Cleaveland, and A. Groce. 2001. *Efficient model checking via buchi tableau automata.* Springer-Verlag, Paris, France, p. 38-52,

13. Bordini, R. H., W. Visser, M. Fisher, C. Pardavila, and M. Wooldridge. 2003. *Model checking multi-agent programs with CASP.* (Computer-Aided Verification): Springer, Berlin, p. 110-113.

14. Bratman, M. 1987. *Intention, plans, and practical reason.* Cambridge, MA, USA: Harvard University Press.

15. Brooks, R. A. 1986. *A robust layered control system for a mobile robot.* IEEE Journal of Robotics and Automation RA-2, (1) (03) p. 14-23.

16. Busetta, P., R. Ronnquist, A. Hodgson, and A. Lucas. 1998. *JACK intelligent agents - components for intelligent agents in java.* Agent Oriented Software Pty. Ltd. Melbouren, Australia,

17. Chirstensen, E., Curbera, F., Meredith, G. and Weerawarana, W. *Web services description language (WSDL)* 1.1. 2001 Available from http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

18. Cleaveland, R. 1990. *Tableau-based model checking in the propositional mu-calculus.* Acta Informatica 27, (8) p 725-47.

19. Cleaveland, R., and S. T. Sims. 2002. *Generic tools for verifying concurrent systems.* Science of Computer Programming 42, (1) p 39-47.

20. Curbera, F., Rania Khalaf, Nirmal Mukhi, S. Tai, and Sanjiva Weerawarana. 2003. *The next step in Web services.* Communications of the ACM 46, (10) (10) p 29-34.

21. Emerson, E. A., C. S. Jutla, and A. P. Sistla. 1993. *On model-checking for fragments of μ-calculus.* Paper presented at 5th International Conference, CAV '93 Proceedings,

22. Fensel, D. 2001. Ontology: *Ontologies and electronic commerce.* IEEE Intelligent Systems, v 16, n 1, Jan.-Feb. 2001, p 8-14

23. Fensel, D., F. Van Harmelen, Y. Ding, M. Klein, H. Akkermans, J. Broekstra, A. Kampman, et al. 2002. *Ontology-based knowledge management.* Computer 35, (11) p 56-9.

24. Finin, T., R. Fritzson, D. McKay, and R. McEntire. 1994. *KQML as an agent communication language.* International Conference on Information and Knowledge Management, Proceedings, 1994, p 456

25. Hindriks, K. V., F. S. De Boer, Wiebe van der Hoek, and J. -J C. Meyer. 1999. *Agent programming in 3APL.* Autonomous Agents and Multi-Agent Systems 2, (4): 357-401.

26. Labrou, Y. 1., and T. 1. Finin. 1997. *Semantics and conversations for an agent communication language.* Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, 1997 *(IJCAI-97)*, vol.1 p 584-91

27. Luck, M., R. Ashri, and M. D'Inverno. 2004. *Agent-base software development.* Boston, London: ArtechHouse.

28. Maes, P. 1990. *Situated agents can have goals.* Robotics and Autonomous Systems 6, (1-2) (06):p 49-70.

29. McBurney, P., and S. Parsons. 2002. *Games that agents play: A formal framework for dialogues between autonomous agents.* Journal of Logic, Language and Information 11, (3): p 315-34.

30. McBurney, P., S. Parsons, and M. Wooldridge. 2002. *Desiderata for agent argumentation protocols.* Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems,(AAMAS-02) 2002, p 402-409

31. Milanovic, N., and M. Malek. 2004. *Current solutions for Web service composition.* IEEE Internet Computing 8, (6): 51-9.

32. Milner, R., *Operational and algebraic semantics of concurrent processes.* J. van Leeuwen, ed., Handbook of Theoretical Computer Science, (Elsevier, Amsterdam, 1990) 1201-1242.

33. Moraitis, P., and N. Spanoudakis. 2007. *Argumentation-based agent interaction in an ambient- intelligence context.* IEEE Intelligent Systems 22, (6) (11) p 84-93.

34. Moreira, A. F., R. Vieira, and R. H. Bordini. 2004. *Extending the operational semantics of a BDI agent-oriented programming language for introducing speech-act based communication.* Paper presented at Revised Selected and Invited Papers,

35. Ouzzani, Mourad, and Athman Bouguettaya. 2004. *Efficient access to Web services.* IEEE Internet Computing 8, (2) p 34-44.

36. Parsons, S., M. Wooldridge, and L. Amgoud. 2003. *Properties and complexity of some formal inter-agent dialogues.* Journal of Logic and Computation 13, (3) (06) p 347-76.

37. Pokahr, A., and L. Braubach. 2007. *Jadex user guide.*
Available from http://vsis-www.informatik.uni-hamburg.de/projects/jadex/download.php.

38. Pokahr, A., L. Braubach, and W. Lamersdor. 2005. *Jadex: A BDI reasoning engine.* In Multi-agent programming, languages, plateforms and applications., eds. R. Bordini, M. Dastani, J. Dix and A. Seghrouchni Springer.

39. Rao, A. S., and M. P. Georgeff. 1992. *An abstract architecture for rational agents.* Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR '92*)*, 1992, p 439-49

40. Rosenschein, J. S., and G. Zlotkin. 1994. *Rules of encounter: Designing conventions for automated negotiation among computer.* Cambridg, Mass: MIT Press.

41. Singh, M. P., and M. N. Huhns. 2005. *Service-oriented computing: Semantics, processes, agents.* John Wiley and Sons.

42. Sirin, E., J. Hendler, and B. Parsia. 2003. *Semi-automatic composition of Web services using semantic descriptions.* Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS.

43. Srivastava, B., and J. Koehler. 2003. *Web service composition -- current solutions and open problems.* In proceeding of International Conference on Automated Planning and Scheduling, Trento, Italy.

44. Tsalgatidou, Aphrodite, and Thomi Pilioura. 2002. *An overview of standards and related technology in Web services.* Distributed and Parallel Databases 12, (2-3): 135-162.

45. UDDI. The UDDI technical white paper. 2000 Available from http://www.uddi.org/.

46. Vardi, M. Y., and P. Wolper. 1986. *An automata-theoretic approach to automatic program verification.* Proceedings of the Symposium on Logic in Computer Science (Cat. No.86CH2321-8), p 332-344.

47. Vermeulen, C., and B. Bauwens. 1998. *Software agents using XML for telecom service modelling: A practical experience.* Proceedings of SGML/XML Europe '98. From Theory to New Practices, p 253-262.

48. W3C. *SOAP 1.2 working draft. 2001* Available from http://www.w3c.org/TR/2001/WD-soap12-part0-20011217.

49. Walton, D.N. and Krabble, E.C.W., 1995. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*, Albany, NY: SUNY press

50. Wooldridge, M. 2002. *An introduction to MultiAgent systems.* John Wiley & Sons.

51. Wooldridge, M. 2000. *Reasoning about rational agents.* Cambridge, MA: The MIT Press.

52. Wooldridge, M., and N. R. Jennings. 1995. *Agent theories, architectures, and languages: A survey.* Intelligent Agents. ECAI-94 Workshop on Agent Theories, Architectures, and Languages Proceedings, 1995, p 1-39

53. Yang, Jian, and M. P. Papazoglou. 2002. Web component: *A substrate for Web service reuse and composition.* Advanced Information Systems Engineering. 14th International Conference, CAiSE. Lecture Notes in Computer Science Vol.2348, p 21-36

54. *Argumentation in multi-agent systems.* Rahwan, I.; Moraitis, P.; Reed, C. eds. First international workshop on Argumentation in Multi-agent systems, ArgMAS. Springer-Verlag,

Appendix I. Tableau Rules for CTL*^{CA}

Table App I.1 Tableau rules for propositional and universal formulas

$$R1\ \wedge: \frac{\psi_1 \wedge \psi_2}{\psi_1\ \psi_2} \quad R2\ \vee: \frac{\psi_1 \vee \psi_2}{\psi_1\ \psi_2} \quad R3\ \vee: \frac{E(\psi)}{\psi}$$

$$R4\ \neg: \frac{\neg\psi}{\psi} \quad R5\ ?: \frac{?\psi}{\psi} \quad R6\ \neg: \frac{A(\Phi)}{E(\neg\Phi)}$$

Note : "?": express the tableau rule of the challenge action
"?ψ": a given agent does not know whether ψ is ture or not.

Table App I.2 Tableau rules for action formulas

$$R7 < C >: \frac{E(\Phi, C(Ag_1, PC(Ag_1, Ag_2, t, \phi)))}{E(\Phi, PC(Ag_1, Ag_2, t, \phi))}$$

$$R8 < W >: \frac{E(\Phi, Withdraw(Ag_1, PC(Ag_1, Ag_2, t, \phi)))}{E(\Phi, \neg PC(Ag_1, Ag_2, t, \phi))}$$

$$R9 < S_{PC}^{Ag1} >: \frac{E(\Phi, Satisfy(Ag_1, PC(Ag_1, Ag_2, t, \phi)))}{E(\Phi, \phi)}$$

$$R10 < V_{PC}^{Ag1} >: \frac{E(\Phi, Violate(Ag_1, PC(Ag_1, Ag_2, t, \phi)))}{E(\Phi, \neg\phi)}$$

$$R11 < Rea >: \frac{E(\Phi, Reactivate(Ag_1, PC(Ag_1, Ag_2, t, \phi)))}{E(\Phi, PC(Ag_1, Ag_2, t, \phi))}$$

$$R12 < Ch >: \frac{E(\Phi, Challenge(Ag_2, PC(Ag_1, Ag_2, t, \phi)))}{E(\Phi, PC(Ag_2, Ag_1, t', ?\phi))}$$

$$R13 < Acc >: \frac{E(\Phi, Accept(Ag_2, PC(Ag_1, Ag_2, t, \phi)))}{E(\Phi, PC(Ag_2, Ag_1, t', \phi))}$$

$$R14 < Ref >: \frac{E(\Phi, Refuse(Ag_2, PC(Ag_1, Ag_2, t, \phi)))}{E(\Phi, PC(Ag_2, Ag_1, t', \neg\phi))}$$

$$R15 < Jus >: \frac{E(\Phi, Justify(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi)))}{E(\Phi, PC(Ag_1, Ag_2, t', \phi' \therefore \phi))}$$

$$R16 < Att >: \frac{E(\Phi, Attack(Ag_2, PC(Ag_1, Ag_2, t, \phi' \therefore \neg\phi)))}{E(\Phi, PC(Ag_2, Ag_1, t', \phi' \therefore \neg\phi))}$$

$$R17 < Def >: \frac{E(\Phi, Defend(Ag_1, PC(Ag_1, Ag_2, t, \phi' \therefore \phi)))}{E(\Phi, PC(Ag_1, Ag_2, t', \phi' \therefore \phi))}$$

Note: label "<C>" is the label associated with the creation action of a propositional commitment PC.

Table App I.3 Tableau rule for propositional commitment formula

$R18\ [PC_{Ag_1}]: \dfrac{E(\Phi, PC(Ag_1, Ag_2, t, \phi))}{E(\Phi, \phi)}$

Table App I.4 Tableau rules for state formulas

$R19\ \langle\equiv\rangle: \dfrac{E(\Phi, l)}{l,\ E(\Phi)}$	$R20\ \wedge: \dfrac{E(\Phi, \phi_1 \wedge \phi_2)}{E(\Phi, \phi_1, \phi_2)}$
$R21\ \vee: \dfrac{E(\Phi, \phi_1 \vee \phi_2)}{E(\Phi, \phi_1)\ E(\Phi, \phi_2)}$	
$R22\ ?: \dfrac{E(\Phi, ?\psi)}{E(\Phi, \psi)}$	
$R23\ X^-: \dfrac{E(\Phi, X^-\phi_1, \ldots, X^-\phi_n)}{E(\Phi, \phi_1, \ldots, \phi_n)}$	$R24\ X^+: \dfrac{E(\Phi, X^+\phi_1, \ldots, X^+\phi_n)}{E(\Phi, \phi_1, \ldots, \phi_n)}$
$R25\ \wedge: \dfrac{E(\Phi, \phi_1 \therefore \phi_2)}{E(\Phi, \phi_1, X^+(\neg\phi_1 \vee \phi_2)}$	
$R26\ \vee: \dfrac{E(\Phi, \phi_1 U^- \phi_2)}{E(\Phi, \phi_2)\ E(\Phi, \phi_1, X^-(\phi_1 U^- \phi_2))}$	$R27\ \vee: \dfrac{E(\Phi, \phi_1 U^+ \phi_2)}{E(\Phi, \phi_2)\ E(\Phi, \phi_1, X^+(\phi_1 U^+ \phi_2))}$

Appendix II. Proof of Lemmas

Proof of Lemma 5.1

The proof is based on the analysis of the different cases of our tableau rules. Most cases are straightforward. Here we only consider rules $R7$, $R25$, and $R27$.

$R = R7$:

$\sigma_1 \rightarrow_R \sigma_2$

$\Rightarrow \sigma_1 = \mathrm{E}(\Phi, C(Ag_1, PC(Ag_1, Ag_2, t, \psi), \sigma_2 = \mathrm{E}(\Phi, SC(Ag_1, Ag_2, t, \psi))$

$\Rightarrow \sigma_1 \prec \sigma_2$ (from the definition of $\prec$ and the fact that

$|C(PC(Ag_1, Ag_2, t, \psi)| = 1 + |PC(Ag_1, Ag_2, t, \psi)|$

$> |PC(Ag_1, Ag_2, t, \psi)|$)

$R = R25$:

$\sigma_1 \rightarrow_R \sigma_2$

$\Rightarrow \sigma_1 = \mathrm{E}(\Phi, \psi_1 \therefore \psi_2), \sigma_2 = \mathrm{E}(\Phi, \psi_1, X^+(\neg\psi_1 \vee \psi_2))$

$\Rightarrow \sigma_1 \prec \sigma_2$ (from the definition of $\prec$ and the fact that

$|\psi_1 \therefore \psi_2| = 1 + |\psi_1| + |X^+(\neg\psi_1 \vee \psi_2)|$

$R = R27$:

$\sigma_1 \rightarrow_R \sigma_2$

$\Rightarrow \sigma_1 = \mathrm{E}(\Phi, \psi_1 U^+ \psi_2), \sigma_2 = \mathrm{E}(\Phi, \psi_2)$ or $\mathrm{E}(\Phi, \psi_1, X^+(\psi_1 U^+ \psi_2))$

$\Rightarrow \sigma_1 \prec \sigma_2$ (from the definition of $\prec$ and the fact that σ_1 , σ_2

□

Proof of Lemma 5.2

Based on the induction of the structure of ψ, most cases are straightforward. Here we only consider the four following cases:

1- $\psi = X\psi_1$, where $\boldsymbol{X} \in \{X^+, X^-\}$.

We have: $\boldsymbol{CL(X\psi_1) = \{X\psi_1\} \cup CL(\psi_1)}$

Therefore: $\boldsymbol{|CL(X\psi_1)| = 1 + |CL(\psi_1)|}$

Then, by using the induction hypothesis, we conclude that: $\boldsymbol{|CL(X\psi_1)| \leq 1 + 2|\psi_1| \leq 2(1 + |\psi_1|)}$

Then, by using Definition 7 we obtain: $\boldsymbol{|CL(X\psi_1)| \leq 2|X\psi_1|}$

2- $\psi = \psi_1 \, U \, \psi_2$, where $U \in \{U^+, U^-\}$.

We have: $CL(\psi_1 \, U \, \psi_2) = \{\psi_1 \, U \, \psi_2\} \cup CL(\psi_1) \cup CL(\psi_2) \cup CL(X(\psi_1 \, U \, \psi_2))$

$= \{\psi_1 \, U \, \psi_2\} \cup CL(\psi_1) \cup CL(\psi_2) \cup \{X(\psi_1 \, U \, \psi_2)\}$

Therefore: $|CL(\psi_1 \, U \, \psi_2)| = 2 + |CL(\psi_1)| + |CL(\psi_2)|$

Then, by using the induction hypothesis and the previous case, we conclude that:

$|CL(\psi_1 \, U \, \psi_2)| \leq 2 + 2|\psi_1| + 2|\psi_2| + |X(\psi_1 \, U \, \psi_2)|$

Then, by using Definition 7 we obtain: $|CL(\psi_1 \, U \, \psi_2)| \leq 2|\psi_1 \, U \, \psi_2|$

3- $\psi = PC(Ag_1, Ag_2, t, \psi_1)$

We have: $CL(PC(Ag_1, Ag_2, t, \psi_1)) = \{PC(Ag_1, Ag_2, t, \psi_1)\} \cup CL(\psi_1)$

Therefore: $|CL(PC(Ag_1, Ag_2, t, \psi_1))| = 1 + |CL(\psi_1)|$

Then, by using the induction hypothesis, we conclude that:

$|CL(PC(Ag_1, Ag_2, t, \psi_1))| \leq 1 + 2|\psi_1|$

Then, by using Definition 7 we obtain: $|CL(PC(Ag_1, Ag_2, t, \psi_1))| \leq 2|PC(Ag_1, Ag_2, t, \psi_1)|$

4- $\psi = C(Ag_1, PC(Ag_1, Ag_2, t, \psi_1))$

We have: $CL(Create(Ag_1, PC(Ag_1, Ag_2, t, \psi_1)))$

$= \{C(PC(Ag_1, Ag_2, t, \psi_1))\} \cup CL(PC(Ag_1, Ag_2, t, \psi_1))$

Therefore: $|CL(C(Ag_1, PC(Ag_1, Ag_2, t, \psi_1)))| = 1 + 2|CL(PC(Ag_1, Ag_2, t, \psi_1))|$

Then, by using the previous case, we conclude that:

$|CL(C(Ag_1, PC(Ag_1, Ag_2, t, \psi_1)))| \leq 1 + 2|PC(Ag_1, Ag_2, t, \psi_1)|$

Then, by using Definition 7 we obtain:

$|CL(C(Ag_1, PC(Ag_1, Ag_2, t, \psi_1)))| \leq 2|C(Ag_1, PC(Ag_1, Ag_2, t, \psi_1))|$

□

Proof of Lemma 5.3

The proof is based on the case analysis of the rule R. Most cases are straightforward. Here we consider the rules $R7$, $R25$, and $R27$.

$R = R7$:

$\sigma_1 \rightarrow_R \sigma_2$

$\Rightarrow E(\Phi, \psi_1) = E(\Phi, C(Ag_1, PC(Ag_1, Ag_2, t, \psi)),$

$E(\Phi, \psi_2) = E(\Phi, PC(Ag_1, Ag_2, t, \psi))$

$\Rightarrow$ (Definition of $CL(C(Ag_1, PC(Ag_1, Ag_2, t, \psi)))$

$CL(\psi_2) \subseteq CL(\psi_1)$

$R = R25$:

$\sigma_1 \rightarrow_R \sigma_2$

$\Rightarrow E(\Phi, \psi_1) = E(\Phi, \psi \therefore \psi'), E(\Phi, \psi_2) = E(\Phi, \psi, X^+(\neg\psi \vee \psi'))$

$\Rightarrow CL(\psi_1) = \{\psi \therefore \psi'\} \cup CL(\psi) \cup CL(X^+(\neg\psi \vee \psi'))$

$\Rightarrow CL(\psi_2) \subseteq CL(\psi_1)$

$R = R27$:

$\sigma_1 \rightarrow_R \sigma_2$

$\Rightarrow E(\Phi, \psi_1) = E(\Phi, \psi\, U^+ \psi'), E(\Phi, \psi_2) = E(\Phi, \psi')$ or $E(\Phi, \psi, X^+(\psi\, U^+\, \psi'))$

$\Rightarrow CL(\psi_1) = \{\psi\, U^+\, \psi'\} \cup CL(\psi) \cup CL(\psi') \cup CL(X^+(\psi\, U^+\, \psi'))$

$\Rightarrow CL(\psi_2) \subseteq CL(\psi_1)$

□

Proof of Lemma 5.4:

Suppose that there exists an infinite chain: $\sigma_1 \prec \sigma_2 \prec \ldots$

From Lemma 3, it follows that $CL(\psi_i) \subseteq CL(\psi_{i-1}) \subseteq \ldots CL(\psi_1)$

Since $CL(\psi_1)$ is finite (from Lemma 2), it follows that:

$\exists j, \forall k \geq j, CL(\psi_k) = CL(\psi_j)$ with $\sigma_j \prec \sigma_{j+1} \prec \ldots \sigma_k \prec \ldots$

However, this is contradictory (from Lemma 3).

□