University of Crete
School of Sciences and Engineering
Department of Computer Science

Automated Negotiation and Semantic Brokering with Intelligent Agents Using Defeasible Logic

Thomas Skylogiannis

Msc Thesis

Heraklio, January 2005

Πανεπιστήμιο Κρήτης
Σχολή Θετικών Επιστημών
Τμήμα Επιστήμης Υπολογιστών

Αυτόματη Διαπραγμάτευση και Σημασιολογική Μεσιτεία με Χρήση Ευφυών Πρακτόρων και Αναιρέσιμης Συλλογιστικής

Θωμάς Σκυλογιάννης

Μεταπτυχιακή Εργασία

Ηράκλειο, Ιανουάριος 2005

Πανεπιστήμιο Κρήτης
Σχολή Θετικών Επιστημών
Τμήμα Επιστήμης Υπολογιστών

Αυτόματη Διαπραγμάτευση και Σημασιολογική Μεσιτεία με Χρήση Ευφυών Πρακτόρων και Αναιρέσιμης Συλλογιστικής

Εργασία που υποβλήθηκε από τον
Θωμά Σκυλογιάννη
ως μερική εκπλήρωση των απαιτήσεων για την απόκτηση
ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ

Συγγραφέας:

_____

Θωμάς Σκυλογιάννης
Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

Εισηγητική Επιτροπή:

_____

Γρηγόρης Αντωνίου Καθηγητής, Επόπτης

_____

Κωνσταντίνος Στεφανίδης Καθηγητής, Μέλος

_____

Αναστασία Αναλυτή Κύρια Ερευνήτρια, Μέλος

Δεκτή:

_____

Δημήτρης Πλεξουσάκης
Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών
Ηράκλειο, Ιανουάριος 2005

Automated Negotiation and Semantic Brokering with Intelligent Agents using Defeasible Logic

Thomas Skylogiannis

Master of Science Thesis

Computer Science Department, University of Crete

Abstract

E-Commerce describes the revolution that is currently transforming the way business is conducted, through the use of information technology, and in particular the World Wide Web. Brokering and negotiation are fundamental stages of e-Commerce. The last few years, there has been an increasing interest for the automation of these two stages. This is partially achieved by the use of software agents. As machines start to engage in business processes, information must be organized in such a way that is accessible by both humans and machines. Additionally, machines must be able to access, process and interpret the information in the same way. Semantic Web vision promises to handle these new issues. Electronic Brokering is a good candidate for taking up Semantic Web technology. We study the brokering or matchmaking problem that is, how a requester's requirements and preferences can be matched against a set of offerings collected by a broker. The proposed solution uses the Semantic Web standard of RDF to represent the offerings, and defeasible logic, which is based on non-monotonic reasoning, for expressing the requirements and preferences. We motivate and explain the approach we propose, and report on a prototypical implementation of a semantic brokering system, exhibiting the described functionality, in JADE multi-agent environment. Expression of negotiation strategies is also of great importance in e-Commerce procedure. We exploit the use of defeasible logic for expressing negotiation strategies. We also implement a negotiating logic-based agent architecture in JADE multi-agent environment.

Αυτόματη Διαπραγμάτευση και Σημασιολογική Μεσιτεία με Χρήση Ευφυών Πρακτόρων και Αναιρέσιμης Συλλογιστικής

Θωμάς Σκυλογιάννης
Μεταπτυχιακή Εργασία
Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης

Περίληψη

Ο όρος «ηλεκτρονικό εμπόριο» χαρακτηρίζει την επανάσταση στον τρόπο που διεξάγονται σήμερα οι επιχειρηματικές δοσοληψίες, μέσα από την χρήση τεχνολογιών πληροφοριών και συγκεκριμένα του παγκόσμιου ιστού. Η εύρεση προϊόντων και εμπόρων καθώς και η διαπραγμάτευση, είναι βασικά στάδια του ηλεκτρονικού εμπορίου. Τα τελευταία χρόνια, διακρίνεται ένα αυξανόμενο ενδιαφέρον για την αυτοματοποίηση αυτών των σταδίων. Αυτό επιτυγχάνεται ως ένα βαθμό με την χρήση εφαρμογών πρακτόρων. Καθώς οι μηχανές αρχίζουν να εισχωρούν στις επιχειρησιακές διεργασίες, οι πληροφορίες θα πρέπει στο εξής να οργανώνονται με τέτοιον τρόπο ώστε να είναι προσπελάσιμες και από τους ανθρώπους και από τις μηχανές. Επιπλέον, οι μηχανές θα πρέπει να μπορούν να προσπελάζουν, να επεξεργάζονται και να ερμηνεύουν την πληροφορία με τον ίδιο τρόπο. Το όραμα του σημασιολογικού ιστού, υπόσχεται να αντιμετωπίσει τα νέα ζητήματα που προκύπτουν. Η ηλεκτρονική μεσιτεία είναι ένας καλός τομέας για την υιοθέτηση τεχνολογιών σημασιολογικού ιστού. Στην παρούσα εργασία μελετούμε αυτό το θέμα, που αφορά το ταίριασμα των απαιτήσεων-προτιμήσεων κάποιου που αιτείται μιας υπηρεσίας, με τις διαφημίσεις αυτών που προσφέρουν την αντίστοιχη υπηρεσία. Η προτεινόμενη λύση χρησιμοποιεί το μοντέλο δεδομένων RDF για την έκφραση των προσφορών και την αναιρέσιμη συλλογιστική, η οποία αποτελεί μη μονοτονική λογική, για την έκφραση των προτιμήσεων. Παρέχουμε τα κίνητρα και εξηγούμε την προσέγγιση μας, και κατόπιν υλοποιούμε την προαναφερθείσα λειτουργικότητα με χρήση της πλατφόρμας πολλαπλών πρακτόρων JADE. Η έκφραση των στρατηγικών διαπραγμάτευσης, είναι επίσης σημαντική για την διαδικασία του ηλεκτρονικού εμπορίου. Ερευνούμε την χρήση αναιρέσιμης συλλογιστικής για την μοντελοποίηση στρατηγικών διαπραγμάτευσης και υλοποιούμε μια προτεινόμενη αρχιτεκτονική πράκτορα που βασίζεται στην λογική με χρήση του JADE.

Ευχαριστίες

Αισθάνομαι την ανάγκη να εκφράσω ένα μεγάλο ευχαριστώ στον επόπτη καθηγητή μου Γρηγόρη Αντωνίου, για την ουσιαστική καθοδήγηση και συμβολή του στην ολοκλήρωση της παρούσας εργασίας.

Επίσης θα ήθελα να ευχαριστήσω τον καθηγητή Κωνσταντίνο Στεφανίδη και την Ερευνήτρια του ΙΤΕ-ΙΠ Αναστασία Αναλυτή, για την συμμετοχή τους στην εξεταστική επιτροπή και για τις πολύτιμες συμβουλές που μου έδωσαν.

Ευχαριστώ από τα βάθη της ψυχής μου τους γονείς μου, Γιάννη και Κατερίνα και την αδελφή μου Σύλβια για την υλική και ψυχική στήριξη που μου πρόσφεραν καθ' όλη την διάρκεια της ακαδημαϊκής μου πορείας.

Τέλος ευχαριστώ όλους τους φίλους μου για τις στιγμές που περάσαμε μαζί και για όλη την βοήθεια που μου πρόσφεραν κατά την διάρκεια των σπουδών μου στο Ηράκλειο της Κρήτης.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation and Contribution of the Thesis

In the last few years, there has been a great interest in *electronic commerce* (e-commerce) potential [1]. As the number of transactions carried out through the Internet increases, the interest for partial or full automation of these transactions increases as well. E-commerce describes the revolution that is currently transforming the way business is conducted, through the use of information technology, and in particular the World Wide Web. In order to understand the basic stages of an e-commerce procedure, we must examine a consumer's buying behaviour model.

Such a model is described in [1]. According to this, there are six basic steps, which appear during the buying process. *Need Identification* is the realization by the buyer, that he needs a particular product. *Product Brokering* is the information gathering by the buyer, so that he is able to decide what to buy. The buyer provides some criteria and a filtering mechanism is employed to provide him with a set of products. *Merchant Brokering* combines the products set from the previous stage with information for specific merchants and gives the buyer a set of potential sellers. *Negotiation is* the stage during which both participants try to reach an agreement, over possible negotiation issues, such as the price, the volume or the delivery time of a product. *Purchase and Delivery* follow the negotiation phase and payment or delivery options are examined. Finally *Product Service and Evaluation* is the evaluation of product and customer service, made by the buyer in order to estimate his utility.

According to [1], in the 1st generation e-commerce applications (current state), buyers and sellers are humans who typically browse through a catalogue of well-defined commodities (e.g. flights, books…) and make fixed price purchases, usually by means of credit card transaction. Humans are in the loop of all stages of buying process, something that is time consuming. Thus, most of the research today is conducted towards the realization of 2nd generation (future state) of e-commerce applications, which will be realized through the use of automated methods of information technology. Web users will be represented by S*oftware Agents*. According to [2], there is an increasing use of software agents for all the aspects of e-Commerce.

However, as software agents start to engage in e-commerce, new issues arise. Information must be organized in a way that is accessible by both humans and machines. Additionally, machines must be able to access, process and interpret the information in the same way. This vision is consistent with the *Semantic Web* initiative [47], which enriches the current Web through the use of machine-processable information about the meaning (or semantics) of information content. This way, the meaning of displayed information is accessible not only to humans, but becomes also accessible to software agents. The key techniques of the Semantic Web are *Semantic Annotations (meta-data*), such that web information carries its meaning on its sleeve, and *Ontologies* which organize terms in a conceptualisation of a domain, thus connecting semantic annotations with each other and serving as a basis for interoperability.

The focus of our work is on the Negotiation and Brokering stages of the e-commerce procedure that we have already described. We use logic-based, agent-based and semantic web-based technologies, in order to deal with these two stages. In the following paragraphs, we describe the motivation for and the contribution of our work, starting with the negotiation part and continuing with brokering.

The basic dimensions of negotiation are *Negotiation Protocols* and *Negotiation Strategies [14]*. *Negotiation protocol* is a set of rules that govern the interaction and *negotiation strategy* is a decision making model, which participants employ, in order to achieve their goal, in line with a negotiation protocol. *We are interested in the negotiation strategy dimension*. One issue in this context is thus “How to express a negotiation strategy”? In addition, there are several techniques for creating negotiation strategies such as game-theoretic, heuristic and argumentation-based. Thus, another issue is: «Is there any framework which can model strategies which are based on all above mentioned techniques”?

Many approaches provide a set of simple pre-defined strategies [8], [20]. They argue that using pre-defined strategies has the advantage that it makes the bidding agents' decisions easily explainable and predictable. However, these approaches considerably restrict the possibilities of the users for expressing their preferences. In the other extreme ([9], [22], [24]), each participant involved in a negotiation must develop his own negotiating agent “from scratch", using a general-purpose design methodology and development environment. Although this approach is unavoidable when very complex strategies are considered, its systematic use leads to time-consuming and hard-to-reuse

development efforts. In addition, agents developed under this approach have static functionalities, in the sense that it is often not possible to modify their behaviour without having to rebuild them from scratch. Lastly there is a plethora of work which exploits and proposes strategies which are based on game-theoretic, heuristic or argumentation-based techniques [4], [7], [15], [19], [21] and [23].

As an alternative to the above two extreme approaches, we propose a simple yet expressive framework for specifying negotiating agents' strategies, in a way that their decisions are predictable and explainable. Specifically, we explore the suitability of defeasible logic programming for expressing the decision-making process of negotiating agents. We built our work on the negotiating agent architecture proposed in [10]. That leads to a plug-and-play negotiation strategy specification. To validate the viability of our approach, we apply it to several negotiation scenarios and provide a prototype implementation of such agent architecture using JADE multiagent framework. We must stress that we do not implement a complete negotiation platform, but rather a fundamental infrastructure, which hosts defeasible logic-based negotiating agents.

When it comes to brokering, its basic dimensions are a framework for expressing *Advertisements* of service providers and a mechanism that captures users' *Preferences* and *Requests*. A broker matches the preferences against the provided advertisements and returns to the service requester the result. In today's state of the art, there are various techniques for performing brokering. According to earlier approaches [29], both advertisements and request are expressed in a logic programming language such as Prolog or LDL. Other systems use feature-based, constraint-based or collaborative techniques [78], [79], [80]. More recent approaches [27], [28], [30], [31], [32] use web ontology languages, such as RDF DAML+OIL and OWL for expressing the advertisements and the queries, or use one of the previous languages for expressing advertisements and a query language like those described in [51] for making requests.

As an alternative to these techniques we propose the use of a hybrid technique for brokering. In particular, advertisements are expressed in a web ontology language (RDF) and user preferences and requests are captured by a declarative logic programming language that is defeasible logic. We demonstrate how our approach is slightly better or at least complementary to the others. Subsequently, we provide a prototype implementation of a brokering system architecture again using the JADE multiagent framework.

1.2 Organization of the Thesis

In section 2 we provide background theory and related work. More specifically, we define the terms “intelligent agent” and “semantic web” and gain a better insight into these issues. Consequently, we present in detail defeasible logic, which is the knowledge representation framework of our work, regarding both brokering and negotiation. We also define the terms “negotiation” and “brokering”, and describe the basic techniques and requirements for such systems. Subsequently we present all the related work in those fields.

In section 3, we justify the eligibility of defeasible logic for expressing negotiation strategies in negotiation cases and user preferences in brokering cases. Subsequently we provide the abstract architecture of our negotiating agent and our brokering system and explain how our work seems to fulfill the posed requirements. Finally we model indicative examples in both fields of negotiation and brokering to prove the viability of our approach.

In section 4 we briefly discuss the multiagent platform we use. Subsequently, we present the architecture of the proposed negotiating agent system. We examine the implementation of a protocol for 1-1 bilateral negotiation that is hard-coded to our agents and express a heuristic-based strategy in defeasible logic. We also present a detailed trace of the negotiation progress. In a later step we present the semantic brokering architecture and create a brokering scenario to demonstrate the operation of the system. Lastly we discuss the system limitations and design conventions.

Finally, we conclude in section 5 and describe possible future work.

2. Background Theory and Related Work

In this section, we firstly refer to the notion of *Intelligent Agents*. We define the term "intelligent agent" and describe its basic attributes. We also enumerate a number of challenges regarding Multi Agent Systems (MAS). Afterwards, we gain a better insight into aspects, such as agent discovery, communication and interaction.

Subsequently, we discuss about the *Semantic Web* initiative and present its layered organization. We also present fundamental semantic web technologies, such as XML, RDF, RDF Schema and OWL.

As a next step we present *Defeasible Logic*. We make a brief introduction to nonmonotonic reasoning and intuitively examine defeasible logic. We then provide a formal definition along with a proof theory.

After we have introduced a basic terminology regarding Agents, Semantic Web and Defeasible Logic we present all the concepts relevant to the field of *Negotiation*. In particular, we define the term "negotiation", outline the basic techniques for building negotiation strategies and make classifications of negotiation protocols, against various dimensions. Then, we briefly cite the fundamental requirements for a negotiation system. Lastly, we present the related work in the field of negotiation.

As a next step, we present the concepts, relevant to the field of *Brokering*, in similar lines, we define the term "brokering" and provide an overview of basic techniques for brokering. We also examine the main requirements for developing a brokering system. Finally we present the related work in the field of brokering.

2.1 Intelligent Agents

As we have already mentioned, agent-based technology provides a perfect means for automating the process of negotiation and brokering. A nice definition of an agent is found in [38]. "Agent is an interactive entity, in a shared with other agents environment, which can perceive and act in a proactive or reactive manner, based on shared knowledge of communication and representation".

Some of the most important attributes, which differentiate agents from other programs, are among others:

- Adaptivity is the ability to learn and improve with experience.
- Autonomy is goal-directedness, proactive and self-starting behaviour.
- Collaborative Behaviour is the ability to work with other agents to achieve a common goal.
- Mobility is the ability to migrate in a self-directed way from one platform host to another.
- Temporal Continuity describes the persistence of identity and state for long periods of time.
- Knowledge-level communication ability characterizes the ability to communicate with language more resembling human-like "speech acts" than typical symbol-level program-to-program protocols.
- Reactivity is the ability to selectively sense and act.
- Personality is the adaptability to different preferences and special requirement of users and lastly.
- Inferential Capability-Intelligence is the ability to act, based on abstract task specifications.

2.1.1 Agent Terminology

In the last few years, a terminology relevant to agent technology has started to be used. In [38], the following terms regarding agent technology are recognized: *Agent Architectures*, *Agent System Architectures*, *Agent Frameworks* and *Agent Infrastructures*. Agent Architectures describe agents as separate entities, consisting of three basic modules named perception, reasoning and action module. Agent System Architectures define the interaction of agents under constraints in a common environment. Agent Frameworks are sets of tools and integrated environments for the development of agents and multiagent systems [67], [70]. Finally, Agent Infrastructures provide means for agent communication and common understandings of various concepts. Their basic components are *Ontologies, Interaction Protocols, Communication Languages* and *Communication Infrastructures.* Ontologies are formal specifications, which describe concepts, their properties and relations in a particular domain of interest. Agent Communication Languages (ACLs) provide agents with means of exchanging information and knowledge [34]. Interaction Protocols are the

rules, which enable agents to reason over the effects of their communications [66]. Finally, Communication Infrastructures provide communication channels among agents.

2.1.2 Multi-Agent Systems and Challenges

According to [40], Multi Agent System (MAS) is a loosely coupled network of problem-solver entities that work together to solve problems beyond the individual capabilities of knowledge of each problem solver. In the same work several challenges and issues regarding the design and implementation of a MAS are presented. To be more specific: (a) how can we describe problems, allocate tasks, and compose the results of separate agents in a MAS? (b) How can agents find each other in an open or more closed environment? (c) How can agents communicate and interact in a meaningful way, and how can heterogeneous agents, constructed by independent users, cooperate? (d) How do agents represent and reason about other agents' actions in order to interact with them? (e) How do we recognize and deal with conflicting interaction among a group of agents? From the above challenges we focus on (b), (c) and (d), that we call *Agent Discovery, Agent Interaction and Communication*, and *Agent Inferential Capability* respectively.

2.1.3 Agent and Service Discovery

Agent Discovery is a way of advertising, managing and finding information about agents' services and capabilities. We can distinguish two different categories of agent discovery mechanisms, called *Centralized Discovery Mechanisms* and *Distributed Discovery Mechanisms* respectively.

2.1.3.1 Centralized Solutions

When it comes to centralized solutions for agent discovery, or in other words "middle agents" according to [41], two different kinds of agents prevail. They are called *Matchmakers,* or *Yellow Pages Services,* and *Facilitators* respectively. We borrow the next two figures from their work.

When it comes to matchmakers, different service providers advertise their capabilities (1) and the matchmaker puts them into a repository. When it is asked for a particular service by a requester (2), it returns to him information about all the available service providers (3).

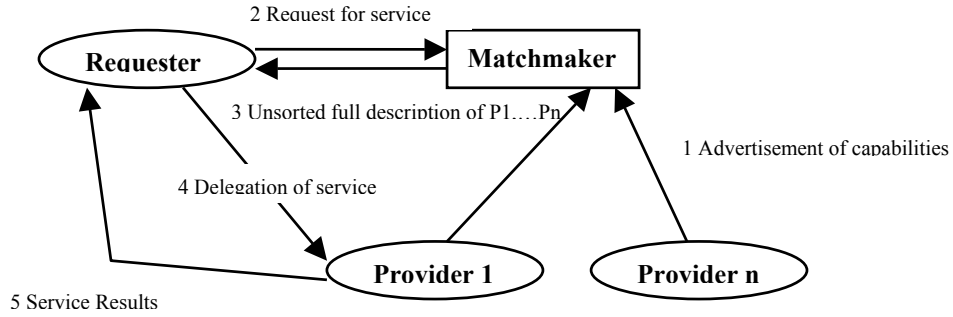


Figure 1. Matchmaker Architecture

It now depends on the requester which provider he will choose (4) for the required service. Lastly, the provider serves the request and returns the results (5). It is assumed that the address of the matchmaker is well known. Facilitators operate in a slightly different way. Initially, providers advertise their capabilities (1). After requesters have located a facilitator, possibly by means of a matchmaker, they give him their preferences along with the delegation of a service (2). The facilitator in turn, picks one of the providers to delegate the requested service (3).

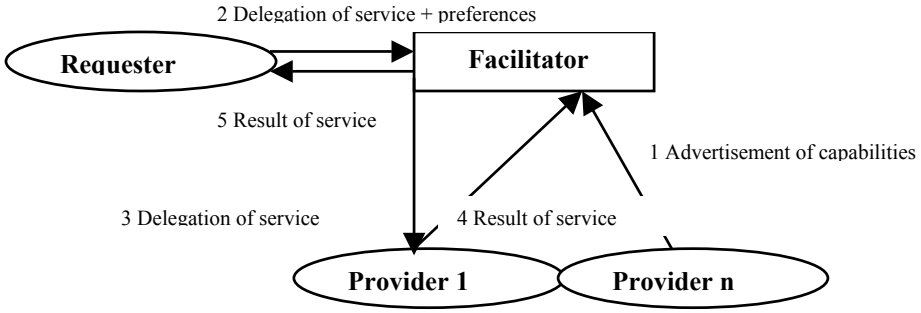


Figure 2. Facilitator Architecture

The provider then returns the result (4) and the facilitator returns it to the requester (5). A *variation* of this architecture could be that the middle agent itself performs the serving of a request using services and information from other agents in conjunction with his own services. In the latter case the middle agent is called “Broker”. We use this variation for our system. So after a service requester has located the Broker by means of a yellow pages service, it can empower it to perform the brokering activity.

2.1.3.2 Distributed Solutions

Distributed solutions harness the power and the abilities of peer-to-peer systems [71] for agent discovery. There are two basic peer-to-peer architectures that are called Hybrid and *Pure* peer-to-peer architectures.

In hybrid peer-to-peer (Napster-like), a large cluster of dedicated central servers maintains an index of the files that are currently being shared by active peers. Each peer maintains a connection to one of the central servers, through which the file location queries are sent. The servers then cooperate to process the query and return a list of matching files and locations. On receiving the results, the peer may choose to initiate a file exchange directly from another peer.

There are no centralized servers in pure peer-to-peer systems (Gnutella-like). Instead, Gnutella peers form an *overlay network* by forging point-to-point connections with a set of neighbors. To locate a file, a peer initiates a controlled flood of the network, by sending a query packet to all of its neighbors. Upon receiving a query packet, a peer checks if any locally stored file matches the query. If so, the peer sends a query response packet back to the query originator. Regardless of whether or not a file match is found, the peer continues to flood the query through the overlay. To help maintain the overlay as the users enter and leave the system, the Gnutella protocol includes *ping* and *pong* messages that allow peers to discover other nodes.

We can find a pure peer-to-peer solution for agent discovery in [37]. The RETSINA project of Carnegie Mellon University, builds upon Gnutella infrastructure for the discovery service employed by the agents. In addition, project JXTA [42] of Sun Microsystems builds upon JXTA core, which provides core support for peer-to-peer services such as naming, creation/deletion policies, advertising, discovery of other peers etc. Project JADE [68], [35] of TILAB uses a hybrid peer-to-peer mechanism for agent discovery.

2.1.4 Agent Communication

The most usual but not unique way for agents to communicate is to send messages encoded by means of an Agent Communication Language (ACL). The most popular ACLs are KQML (Knowledge Query and Manipulation Language), described in [33] and FIPA (Foundation for Intelligent Physical Agents), which is described in [66]. The types of ACL messages are called speech-acts (e.g. INFORM, QUERY, PROPOSE…).

The use of an ACL results in loosely coupled open systems, which only use message passing for collaboration and operation. Protocols such as TCP/IP, IIOP or HTTP are used for message (ACL message) transport.

Other methods for exchange of information between agents are RPC/RMI and CORBA technologies. According to [34], the advantages of ACLs over the above methods for agent communication are: a) ACLs handle propositions, rules and actions instead of simple objects with no semantics associated with them b) the ACL message describes a desired state in a declarative language rather than a procedure or method.

ACL messages are built up of three layers of languages [36]. Elements in the world are defined in an ontology. An agent's intention to describe or alter the world is expressed by a speech-act such as INFORM, and lastly statements about the world are expressed by means of a *Content Language*. The most usual content languages are SL, [76] and its variations, which are FIPA standards, and FIPA-RDF [77], which is still in an experimental stage, although there is an increasing interest for it the last few years.

In order for agents to be able to reason about the effects of their communication, ACL messages should be inserted into proper Agent Interaction Protocols [72] like FIPA propose, FIPA contract net etc. Interaction Protocol describes allowed sequences of actions among agents.

Message Content Ontology (M.C.O.) is a special purpose ontology, which helps agents to describe facts, beliefs, hypotheses and predications about a domain. A M.C.O is usually based on a Content Reference Model (C.R.M), which is derived from the semantic of an ACL and requires content expressions to have proper characteristics according to the message performative. (INFORM…). The elements of a Content Reference Model can be, for example, *Concepts* which are entities that exist in the world, *Predicates* which express the status of part of the world and must the content of a INFORM message, *Agent Actions* which express the action an agent can request from another agent and must be the content of a REQUEST message and finally Identifying Referential Expression (IRE) which, identify entities for which a given predicate is true. IREs must be the content of a QUERY_REF message.

2.2 Semantic Web

The aim of the Semantic Web initiative is to advance the state of the current Web through the use of semantics. More specifically, it proposes to use *semantic*

annotations to describe the meaning of certain parts of Web information. For example, the Web site of a hotel could be suitably annotated to distinguish between hotel name, location, category, number of rooms, available services etc. Such meta-data could facilitate the automated processing of the information on the Web site, thus making it accessible to machines and not primarily to human users, as it is the case today.

However, the question arises as to how the semantic annotations of different Web sites can be combined, if everyone uses terminologies of their own. The solution lies in the organization of vocabularies in so-called *ontologies*. References to such shared vocabularies allow interoperability between different Web resources and applications. For example, an ontology of hotel classifications in a given country could be used to relate the rating of certain hotels. And a geographic ontology could be used to determine that Crete is a Greek island and Heraklion a city on Crete. Such information would be crucial to establish a connection between a requester looking for accommodation on a Greek island, and a hotel advertisement specifying Heraklion as the hotel location.

The development of the Semantic Web proceeds in steps, each step building a layer on top of another. The layered design is shown in Fig. 3, which is outlined below.

- At the bottom layer we find *XML*, a language that lets one write structured web documents with a user-defined vocabulary. XML is particularly suitable for sending documents across the Web, thus supporting syntactic interoperability.
- *RDF* is a basic data model, like the entity-relationship model, for writing simple statements about Web objects (resources). The RDF data model does not rely on XML, but RDF has an XML-based syntax. Therefore, it is located on top of the XML layer.
- *RDF Schema* provides modeling primitives, for organizing Web objects into hierarchies. RDF Schema is based on RDF. RDF Schema can be viewed as a primitive language for writing ontologies.
- But there is a need for more powerful *ontology languages* that expand RDF Schema and allow the representations of more complex relationships between Web objects. Ontology languages, such as OWL, are built on the top of RDF and RDF Schema.

- The *logic layer* is used to enhance the ontology language further, and to allow writing application-specific declarative knowledge.
- The *proof layer* involves the actual deductive process, as well as the representation of proofs in Web languages and proof validation.
- Finally *trust* will emerge through the use of digital signatures, and other kind of knowledge, based on recommendations by agents we trust, or rating and certification agencies and consumer bodies.

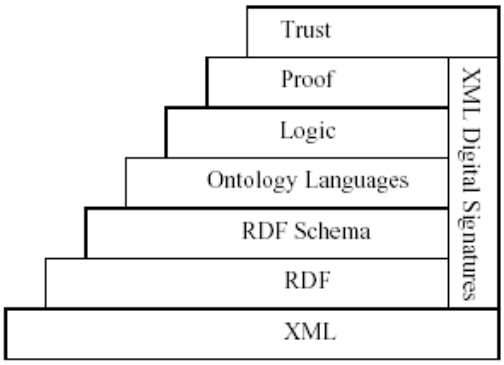


Figure 3. The Semantic Web Tower

For an easy, yet comprehensive introduction to the Semantic Web refer to [45]. In the following subsections we describe XML, RDF, RDF Schema and Ontology layers a bit further. In addition, the logic layer will be the content of section 2.3. Lastly as proof and trust layers are totally out of the scope of this thesis, we choose not to describe them further.

2.2.1 XML Basic Features

XML [48], stands for eXtensible Markup Language. It is a mark-up language much like HTML. XML was designed to describe data and its tags are not predefined. The user must define his own tags by using a Document Type Definition (DTD) or an XML Schema to define the legal building blocks of an XML document, that is, define elements and attributes that can appear in a document, which elements are child elements, what is the order of child elements etc. XML with a DTD or XML Schema is self-descriptive. We must say that XML is not a replacement for HTML. They were designed with different goals: The former was designed to describe data and to focus on what data is and the latter was designed to display data and to focus on how data

looks. HTML is about displaying information, while XML is about describing information. XML was created to structure, store and share information.

2.2.2 RDF Basic Features

RDF [46] stands for Resource Description Framework and its purpose is to describe resources on the Web. RDF is designed to be read by computers. The basic RDF data model consists of three fundamental concepts: *Resources*, *Properties* and *Statements*.

Resources are the central concept of RDF and are used to describe individual objects of any kind, for example Web pages, people, hotels, flights etc. Every resource has a URI, a Universal Resource Identifier, which can be a Web address or some other kind of unique identifier.

Properties express specific aspects, characteristics, attributes, or relations between resources. For example, properties might be the number of rooms in a hotel, proximity to the beach etc.

Finally statements are composed of a specific resource, together with a named property and the *value* of that property for that resource. The value can be a resource in turn; for example, the manager of Agapi Beach hotel is Alexis Zorbas. Alternatively, the value can be a *literal*, a primitive term that is not evaluated by an RDF processor. For example, the number of rooms of Agapi Beach is 117.

A statement consists of three parts (subject, predicate, object) and is often referred to as an RDF triple. A triple of the form (x, P, y) corresponds to the logical formula P(x, y), where the binary predicate P relates the object x to the object y; this representation is used in our system for translating RDF statements into a logical language ready to be processed automatically in conjunction with rules.

Another possible representation is the graph based. The graph is directed with labeled nodes and arcs. The arcs are directed from the resource (the subject of the statement) to the value (the object of the statement); see Fig. 4. This kind of graph is known as a *Semantic Net* in the artificial intelligence community.

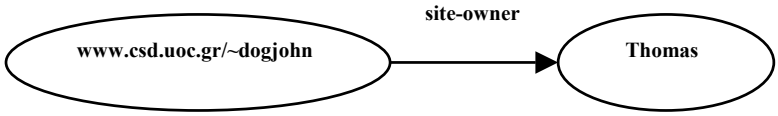


Figure 4. Graph-Based Representation of an RDF Statement

Lastly, there is a third representation based on XML. This representation is compatible with the layered design of the Semantic Web, and facilitates exchange of RDF information among applications. Such a representation is depicted in Fig. 5.

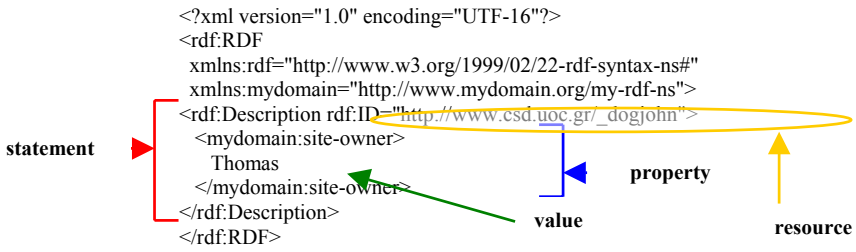


Figure 5. XML-Based Representation of an RDF Statement

2.2.3 RDF Schema Basic Features

RDF is domain-independent, in that no assumptions about a particular domain of use are made. It is up to the users to define their own terminology in a schema language called RDF Schema (RDFS) [49]. In doing so, they actually define a simple ontology, a conceptual model of the domain at hand. The basic features of RDF Schema are the following.

In RDF, Web resources are individual objects. In RDFS, objects sharing similar characteristics are put together to form *classes*. Examples for classes are hotels, airlines, employees, rooms, excursions etc. Individuals belonging to a class are often referred to as *instances* of that class. For example, John Smith could be an instance of the class of employees of a particular hotel.

Binary *properties* are used to establish connections between classes. For example, a property *works_for* establishes a connection between employees and companies. Properties apply to individual objects (instances of the classes involved) to form RDF statements, as seen above.

The application of predicates can be restricted through the use of *domain and range restrictions*. For example, we can restrict the property *works_for* to apply only to employees (domain restriction), and to have as value only companies (range restriction). This way, nonsensical statements due to user errors, for example that Crete works for Agapi Beach, can be automatically detected.

Classes can be put together in hierarchies through the *subclass relationship*: a class C is a subclass of a class D if every instance of C is also an instance of D. For example, the class of island destinations is a subclass of all destinations: every instance of an island destination (e.g. Crete) is also a destination.

The hierarchical organization of classes is important due to the notion of *inheritance*: once a class C has been declared a subclass of D, every known instance of C is *automatically* classified also as instance of D. This has far-reaching implications for matching customer preferences to service offerings. For example, a customer may wish to make holidays on an Indonesian island. On the other hand, the hotel Noosa Beach advertises its location to be Bali. It is not necessary (nor is it realistic) for the hotel to add information that it is located in Indonesia and on an island; instead, this information is inferred by the ontology automatically.

2.2.4 OWL Basic Features

OWL (Web Ontology Language) [50] comes to fill the missing features of RDF and RDFS. According to [45], OWL deals with the following issues that RDF cannot express:

- Local scope of properties: rdfs:range defines the range of a property, say teaches, for all classes. Thus in RDF Schema we cannot declare range restrictions that apply to some classes only. For example, we cannot say that cows eat only plants, while other animals may eat meat, too.
- Disjointness of classes: Sometimes we wish to say that classes are disjoint. For example, male and female are disjoint. But in RDF Schema we can only state subclass relationships, e.g. female is a subclass of person.
- Boolean combinations of classes: Sometimes we wish to build new classes by combining other classes using union, intersection and complement. For example, we may wish to define the class person to be the disjoint union of the classes male and female. RDF Schema does not allow such definitions.
- Cardinality restrictions: Sometimes we wish to place restrictions on how many distinct values a property may or must take. For example, we would like to say that a person has exactly two parents, and that a course is taught by at least one lecturer. Again such restrictions are impossible to express in RDF Schema.

- Special characteristics of properties: Sometimes it is useful to say that a property is transitive (like “greater than”), unique (like “is mother of”), or the inverse of another property (like “eats” and “is eaten by”).

2.3 Defeasible Logic

2.3.1 Nonmonotonic Reasoning

One of the issues that have recently attracted the concentration of the developers of the Semantic Web is the nature of the rule systems that should be employed in the logic layer of the Semantic Web tower. Monotonic rule systems have already been studied and accepted as an essential part of the layered development of the Semantic Web. Nonmonotonic rule systems, on the other hand, seem also to be a good solution, especially due to their expressive capabilities.

Nonmonotonic reasoning is a subfield of Artificial Intelligence trying to find more realistic formal models of reasoning than classical logic. In common sense reasoning one often draws conclusions that have to be withdrawn, when further information is obtained. Thus, the set of conclusions does not grow monotonically with the given information. The latter phenomenon, nonmonotonic reasoning methods try to formalize.

In a monotonic logic system, given a collection of facts D that entail some sentence *s* (*s* is a logical conclusion of *D*), for any collection of facts *D'* such that $D \subseteq D'$, *D'* also entails *s*. In other words: *s* is also a logical conclusion of any superset of *D*.

In a nonmonotonic system, the addition of new facts can reduce the set of logical conclusions. So, if *s* is a logical conclusion of *D*, it is not necessarily a conclusion of any superset of *D.* Two of the basic characteristics of nonmonotonic systems are: *adaptability* (ability to deal with a changing environment), and the ability to reason under conditions of *uncertainty*. In other words, such systems are capable of adding and retracting beliefs as new sets of information is available, and reasoning with an incomplete set of facts.

Defeasible logic, which was introduced by Donald Nute [63] is a representative language of nonmonotonic reasoning. In general, a defeasible theory (a knowledge base in defeasible logic) consists of five different kinds of knowledge: facts, strict rules, defeasible rules, defeaters, and a superiority relation.

Facts are indisputable statements, for example, “Tweety is an emu”. Written formally, this would be expressed as:

```
emu (tweety)
```

Strict Rules are rules in the classical sense: whenever the premises are indisputable (e.g., facts) then so is the conclusion. An example of a strict rule is "Emus are birds". Written formally:

```
emu (X) → bird (X)
```

Defeasible rules are rules that can be defeated by contrary evidence. An example of such a rule is "Birds typically fly"; written formally:

```
bird (X) ⇒ flies (X)
```

The idea is that if we know that something is a bird, then we may conclude that it flies, *unless there is other, not inferior, evidence suggesting that it may not fly*.

Defeaters are rules that cannot be used to draw any conclusions. Their only use is to prevent some conclusions. In other words, they are used to defeat some defeasible rules by producing evidence to the contrary. An example is "If an animal is heavy then it might not be able to fly". Formally:

```
heavy (X) ⇝ ¬flies (X)
```

The main point is that the information that an animal is heavy is not sufficient evidence to conclude that it does not fly. It is only evidence that the animal *may* not be able to fly. In other words, we do not wish to conclude $\neg flies\ (X)$ if $heavy\ (X)$; we simply want to prevent a conclusion $flies\ (X)$.

The *superiority relation* among rules is used to define priorities among rules, i.e., where one rule may override the conclusion of another rule. For example, given the defeasible rules

```
r: bird (X) ⇒ flies (X)
s: brokenWing (X) ⇒¬flies (X)
```

which contradict one another, no conclusive decision can be made about whether a bird with broken wings can fly. But if we introduce a superiority relation > with $s > r$, with the intended meaning that s is strictly stronger than r, then we can indeed conclude that the bird cannot fly.

Notice that a cycle in the superiority relation is counterintuitive. In the above example, it makes no sense to have both $r > s$ and $s > r$. Consequently, we focus on cases where the superiority relation is acyclic.

Another point worth noting is that, in Defeasible Logic, priorities are *local* in the following sense: two rules are considered to be competing with one another only if they have complementary heads. Thus, since the superiority relation is used to resolve

conflicts among competing rules, it is only used to compare rules with complementary heads; the information $r > s$ for rules r, s without complementary heads may be part of the superiority relation, but has no effect on the proof theory.

A more formal definition of Defeasible Logic and a proof theory are given in the next section.

2.3.2 Formal Definition

In this thesis we restrict attention to essentially propositional Defeasible Logic. Rules with free variables are interpreted as rule schemas, that is, as the set of all ground instances. If q is a literal $\sim q$ denotes the complementary literal (if q is a positive literal p then $\sim q$ is $\neg p$; and if q is $\neg p$, then $\sim q$ is p).

Rules are defined over a *language* (or *signature*) Σ, the set of propositions (atoms) and labels that may be used in the rule. In cases where it is unimportant to refer to the language of D, Σ will not be mentioned.

A *rule* r: $A(r) \hookrightarrow C(r)$ consists of its unique *label* r, its *antecedent* $A(r)$ ($A(r)$ may be omitted if it is the empty set) which is a finite set of literals, an arrow $\hookrightarrow$ (which is a placeholder for concrete arrows to be introduced in a moment), and its *head* (or *consequent*) $C(r)$ which is a literal. In writing rules we omit set notation for antecedents, and sometimes we omit the label when it is not relevant for the context. There are three kinds of rules, each represented by a different arrow. Strict rules use $\rightarrow$, defeasible rules use $\Rightarrow$, and defeaters use $\rightsquigarrow$.

Given a set R of rules, we denote the set of all strict rules in R by Rs, the set of strict and defeasible rules in R by Rsd , the set of defeasible rules in R by Rd , and the set of defeaters in R by $Rdft$. $R[q]$ denotes the set of rules in R with consequent q.

A *superiority relation on* R is a transitive relation $>$ on R. When $r1 > r2$, then $r1$ is called *superior* to $r2$, and $r2$ *inferior* to $r1$. Intuitively, $r1 > r2$ expresses that $r1$ overrules $r2$, should both rules be applicable. Typically we assume $>$ to be acyclic (that is, the transitive closure of $>$ is irreflexive).

A *defeasible theory* D is a triple $(F, R, >)$ where F is a finite set of literals (called *facts*), R a finite set of rules, and $>$ an acyclic superiority relation on R. D is called *decisive* if the atom dependency graph of D is acyclic.

2.3.3 Proof Theory

A *conclusion* of D is a tagged literal and can have one of the following four forms:

- $+\Delta\, q$ which is intended to mean that q is definitely provable in D.
- $-\Delta\, q$ which is intended to mean that we have proved that q is not definitely provable in D.
- $+\partial\, q$ which is intended to mean that q is defeasibly provable in D.
- $-\partial\, q$ which is intended to mean that we have proved that q is not defeasibly provable in D.

If we are able to prove q definitely, then q is also defeasibly provable. This is a direct consequence of the formal definition below. It resembles the situation in, say, default logic: a formula is sceptically provable from a default theory $T = (W, D)$ (in the sense that it is included in each extension) if it is provable from the set of facts W.

Provability is defined below. It is based on the concept of a *derivation* in $D = (F,R,>)$. A derivation is a finite sequence $P = (P(1),\ldots,P(n))$ of tagged literals satisfying the following conditions ($P(1..i)$ denotes the initial part of the sequence P of length i):

$+\Delta$: *If* $P(i + 1) = +\Delta\, q$ *then either*

$q \in F$ *or*

$\exists r \in Rs[q]\ \forall a \in A(r): +\Delta\, a \in P(1..i)$.

Figure 6. Definite Provability in Defeasible Logic

That means, to prove $+\Delta\, q$ we need to establish a proof for q using facts and strict rules only. This is a deduction in the classical sense - no proofs for the negation of q need to be considered (in contrast to defeasible provability below, where opposing chains of reasoning must be taken into account, too).

To prove $-\Delta\, q$, i.e., that q is not definitely provable; q must not be a fact. In addition, we need to establish that every strict rule with head q is *known to be* inapplicable. Thus for every such rule r there must be at least one antecedent a for which we have established that a is not definitely provable ($-\Delta\, a$, Fig. 7).

$-\Delta$: *If* $P(i + 1) = -\Delta\, q$ *then*

$q \notin F$ *and*

$\forall r \in Rs[q]\ \exists a \in A(r): -\Delta\, a \in P(1..i)$.

Figure 7. Definite Non-provability in Defeasible logic

It is worth noticing that this definition of nonprovability does not involve loop detection. Thus if D consists of the single rule $p \rightarrow p$, we can see that p cannot be proven, but Defeasible Logic is unable to prove $-\Delta p$.

$+\partial$: *If* $P(i+1) = +\partial q$ *then either*

(1) $+\Delta q \in P(1..i)$ *or*

(2) (2.1) $\exists r \in Rsd[q]\ \forall a \in A(r): +\partial a \in P(1..i)$ *and*

(2.2) $-\Delta \sim q \in P(1..i)$ *and*

(2.3) $\forall s \in R[\sim q]$ *either*

(2.3.1) $\exists a \in A(s): -\partial a \in P(1..i)$ *or*

(2.3.2) $\exists t \in Rsd[q]$ *such that*

$\forall a \in A(t): +\partial a \in P(1..i)$ *and* $t > s$.

Figure 8. Defeasible Provability in Defeasible Logic

To show that q is provable defeasibly ($+\partial q$ Fig. 8) we have two choices: (1) We show that q is already definitely provable; or (2) we need to argue using the defeasible part of D as well. In particular, we require that there must be a strict or defeasible rule with head q, *which* can be applied (2.1). But now we need to consider possible "attacks", i.e., reasoning chains in support of $\sim q$. To be more specific: to prove q defeasibly we must show that $\sim q$ is not definitely provable (2.2). Also (2.3) we must consider the set of all rules which are not known to be inapplicable and which have head $\sim q$. Essentially each such rule s attacks the conclusion q. For q to be provable, each such rule s must be counterattacked by a rule t with head q with the following properties: (i) t must be applicable at this point, and (ii) t must be stronger than s. Thus each attack on the conclusion q must be counterattacked by a stronger rule.

The definition of the proof theory of Defeasible Logic is completed by the condition $-\partial$. It is nothing more than a strong negation of the condition $+\partial$.

$-\partial$: *If* $P(i+1) = -\partial q$ *then*

(1) $-\Delta q \in P(1..i)$ *and*

(2) (2.1) $\forall r \in Rsd[q]\ \exists a \in A(r): -\partial a \in P(1..i)$ *or*

(2.2) $+\Delta \sim q \in P(1..i)$ *or*

(2.3) $\exists s \in R[\sim q]$ *such that*

(2.3.1) $\forall a \in A(s): +\partial a \in P(1..i)$ *and*

(2.3.2) $\forall t \in Rsd[q]$ *either*

$\exists a \in A(t): -\partial a \in P(1..i)$ *or* $t \not\succ s$.

Figure 9. Defeasible Non-provability in Defeasible Logic

To prove that q is not defeasibly provable, we must first establish that it is not definitely provable. Then we must establish that it cannot be proven using the defeasible part of the theory. There are three possibilities to achieve this: either we have established that none of the (strict and defeasible) rules with head q can be applied (2.1); or $\sim q$ is definitely provable (2.2); or there must be an applicable rule s with head $\sim q$ such that no possibly applicable rule t with head q is superior to s (2.3).

The elements of a derivation P in D are called *lines* of the derivation. We say that a tagged literal L is *provable* in $D = (F, R, >)$, denoted $D \vdash L$, if there is a derivation in D such that L is a line of P. When D is obvious from the context we write $\vdash L$.

2.4 Negotiation

We have already asserted that one of our interests is the *negotiation* stage of e-commerce procedure. Thus, we need some definitions and clarifications of the term "negotiation". Governatori et al. state that:" With negotiation we mean a process involving at least two parties aimed at reaching an agreement that is acceptable by the parties involved" [58]. Jennings et al. [14], give a more formal definition: "Negotiation can be viewed as a distributed search through a space of potential agreements"

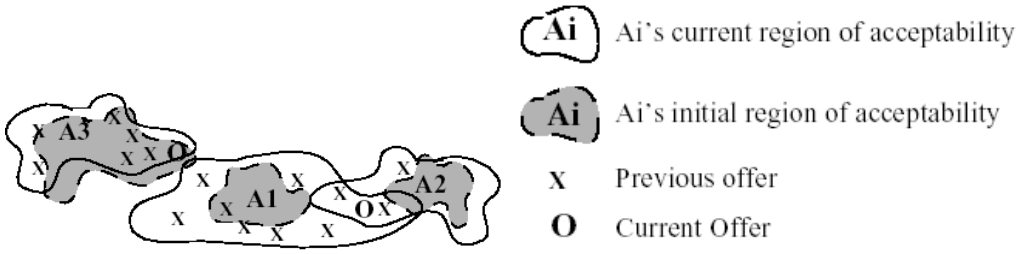


Figure 10. The Space of Negotiation Agreements

When the negotiation starts each participant has a portion of space where he is willing to make agreements (Fig. 10). He also has a way to evaluate the points of his space and some means to determine the agreements he makes by using his evaluation. During the negotiation each participant's space may expand or contract due to external changes or because another participant persuade them to do so. The negotiation ends when

participants find a mutually acceptable point in the negotiation space, which of course belongs in both participants' region of acceptability.

The basic dimensions of negotiation are negotiation protocols and negotiation strategies. *Negotiation protocol* is a set of rules which govern the interaction and a *Negotiation Strategy* is a decision making model, which participants employ in order to achieve their goal in line with the negotiation protocol.

2.4.1 Negotiation Techniques

Jennings et al. [14] identify three different techniques or approaches to express and model negotiation strategies: *Game-theoretic*, *Heuristic* and *Argumentation-based.* We add another one, based on Multi-Objective Decision Analysis (MODA) and Constraint Satisfaction Problems (CSP).

Game-theoretic techniques model the negotiation as a game and apply game theory techniques, to find dominant strategies. Although game theory is a neat solution and the properties of the results are provable, there are many problems with this approach as it requires rational behavior by both players, a lot of computational resources and in many times perfect knowledge of opponents characteristics such as his utility function or his reservation price.

Heuristic techniques use a number of tactics and a strategy, which is a combination of those tactics for the expression of the preferences of the users. The tactics are simple functions that are used to generate an offer, or counter offer, based on different criteria. A strategy is the way in which an agent changes the weights of the different tactics over time. In the negotiation model agents propose offers sequentially following their strategies. Each agent has a scoring function that is used to rate the offers received. If an agent receives an offer that has value greater than the value of the counter offer that he is ready to submit in the next step, then the offer is accepted. Otherwise, the counter offer is submitted. The negotiation tactics include time-dependent tactics, resource-dependent tactics and behavior-dependent tactics.

Argumentation-based techniques are an extension of heuristic approaches and use communication performatives such as lies threats, promises and rewards during the negotiation.

Lastly, there are approaches based on Multi-Objective Decision Analysis (MODA) and Constraint Satisfaction Problems (CSP) [13] prescribes theories for quantitatively

analyzing important decisions involving multiple, interdependent objectives from the perspective of a single decision-maker. This analysis involves two distinctive features: an uncertainty analysis and a utility (i.e., preference) analysis. Techniques such as Bayesian network modeling aid uncertainty analysis. Multi-attribute utility theory (MAUT) analyzes preferences with multiple attributes. MAUT analyzes decision problems *quantitatively* through utilities. CSPs analyze decision problems more q*ualitatively* through constraints. A CSP is formulated in terms of variables, domains, and constraints. Once a decision problem is formulated in this way, a number of general-purpose (and powerful) CSP techniques can analyze the problem and find a solution.

2.4.2 Generic Classification of Protocols for Negotiation

In an attempt to classify negotiation protocols we found several classification criteria, but we don't claim completeness. In particular a negotiation protocol can be classified by:

- The *number of participants*, so there are 1:1, 1:N and M:N negotiations. 1:1 negotiation is called *simple bargaining*, 1:N is called *auction* and finally M:N is called *market.*
- *Bid privacy.* Using this criterion, there are *open-cry* negotiations*,* where buyers and sellers continuously announce their offers, and *sealed-bid* negotiations where the bid of the other participants is kept secret.
- *The number of items to be negotiated.* There are single item and multiple item negotiations. Items can be discussed simultaneously or one by one. The ordering is called the negotiation agenda. If the agenda is pre-defined (before the start of negotiation) it is called exogenous and if it is defined during the negotiation it is called endogenous.
- A*ttributes of negotiated items.* A negotiation can be single-attribute or multi-attribute. During single-attribute negotiation only one aspect, usually the price is negotiated, contrary to multi-attribute negotiations where many aspects such as, delivery time, price, guaranties and delivery options are discussed. The latter is also called integrative negotiation. Example of multi-attribute negotiations is [7].

- *Number of suppliers.* When the potential buyer buys the whole quantity of a product from only one seller this is called *single sourcing* [7]. In *multi-sourcing* the buyer is willing to accept partial quantities from many suppliers. The sum of the partial quantities forms the desired quantity.
- *Degree of automation*. There are semi-automatic and fully automatic negotiations. Semi-automatic negotiations require often feedback from the agent to his owner (monitors). In fully automatic the agents is loaded with a strategy and operates in an autonomous fashion in line with owner's preferences.
- Finally the negotiation can be direct, meaning that buyers and sellers exchange messages in one-step or mediated/brokered, meaning that someone else interferes between them.

2.4.2.1 Competitive vs. Cooperative Protocols for Negotiation

A very interesting aspect of a negotiation protocol is the degree of competitiveness. Competitive negotiation is the conflict of one or two parties over a mutually exclusive goal. On the other hand, cooperative negotiation is the conflict over many interdependent but not mutually exclusive goals [13].

2.4.2.2 Auctions

When there are many buyers and one seller, the negotiation is called *Auction* and when there are many sellers and one buyer the negotiation is called *Reverse Auction*. Below we present the characteristic of the most common auction protocols.

In an *English Auction*, the seller firstly sets a starting price (the *lowest* acceptable price), which either is published or remains secret. He also poses time constraint that is either a time threshold or a maximum accepted time period betweens two subsequent bids. During the auction, offers *increase* by an amount of money above a threshold. When time constraints are violated, the item is knocked down to the bidder who made the last offer [11], [18]. One variation is the open-exit English auction in which the prices rise continuously, but players must publicly announce that they are dropping out when the price is too high [3].

In a *Dutch Auction*, bidding starts at an extremely *high* price and is progressively *lowered* until a buyer claims an item by calling "mine", or by pressing a button that stops an automatic clock [3].

In a *First-price Sealed Bid Auction*, participants submit their bids secretly. No one knows the bid of the others. The participant with the higher bid buys the product and pays the amount he offered [3].

Lastly, in a *Second-price Sealed Bid* (or Vickrey) Auction, something interesting happens. The item is knocked down to the bidder with the highest offer, but he pays a price equal to the second highest bid [3].

2.4.3 Fundamental Requirements for a Negotiation Platform

The requirements for a generic negotiation platform are gracefully defined in [5] and [6]. The existence of a messaging middleware is of great importance. This will enable participants to locate each other and exchange messages. Once the latter is ensured, a framework also needs to define:

- A general protocol for negotiation that can be parameterized with different negotiation rules. Depending on the choice of rules, different negotiation mechanism can occur. The protocol must be sufficiently formal so that automated entities can interact with it. In addition, it must support negotiation both for simple and for complex objects. Lastly to support security mechanisms, for doing business in a trusted way and allow but not require a third person to arbitrate a given negotiation (e.g. an auctioneer in an auction).
- A taxonomy of rules for negotiation. These rules cast the general protocol into one that can be used to embody a particular market mechanism.
- A language to define rules of negotiation. The idea is to have a declarative language for expressing rules in a way that negotiation participants can reason over them. The declarative layer would then be mapped to reusable software components implementing the logic expressed by the rules. These components would be plugged into the orchestration infrastructure for the protocol to be cast to embody a desired market mechanism.

- A language to express negotiation proposals. Such a language may support ontologies and namespaces, exhibit high degree of expressiveness, can express constraints over ranges of possible values as well as definite values of a specification, provide a loose support of types and inheritance, support for complex queries and finally support for complex matching.

2.4.4 An Abstract Negotiation Process

Negotiation takes place by parties communicating via a negotiation locale, which is an abstraction of the used messaging system to be able to negotiate with one another; parties must initially share a negotiation template. The latter specifies the different parameters of the negotiation like price, product type, supply date etc. Some parameters may be constraint while others are subject to negotiation. A negotiation locale has a negotiation template associated with it. As part of the admission process of participants to the negotiation, they are requested to accept the agreement template. The admission step might be formalized by the specification of *admission policies*. The admission policies can specify what credentials (if any) are requested from participants for them to be admitted to the negotiation. The process of negotiation is the move from a *negotiation template* to a *negotiation contract*, which the agreeing parties find acceptable. In the process of reaching agreements, the negotiation participants exchange *proposals,* representing the deals that are currently acceptable to them. Each proposal will contain constraints over some of the parameters, expressed in the agreement template.

2.4.5 Selected Work in Negotiation

Kasbah [8] is an automated negotiation *system*. Users can create buyer or seller agents and engage in a negotiation. Each agent is provided with some information trough a graphical user interface. Such information includes the date and time within which the agent must buy or sell the item, the desired or reservation price, the lowest (for sellers) or highest (for buyers) price he intends to offer and the strategy the agent will follow. There are three predefined strategies, which alter the price in linear, quadratic or exponential manner respectively. Negotiation in Kasbah is bilateral and competitive. When agents find each other the negotiation starts. Kasbah also provides a reputation mechanism, which allows the buyers or sellers to rate opponent's behavior during the negotiation.

[15] Proposes an *approach, which* is based on *argumentation-based* techniques for negotiation. Although other related frameworks are based on concepts in argumentation about beliefs, the authors are based on argumentation about goals. Each agent's knowledge base consists of the following components:

- A set of goals the agent poses.
- A set of beliefs, which can be seen as the context, in which a goal holds.
- A set of statements of the form justify(B,g). These statements link a goal (g) to a set of beliefs (B) that form the justification of a goal.
- A set of statements of the form achieve(subG,G). These statements denote the relation between a goal g and the set subG of subgoals that need to be achieved in order for g to be achieved.
- A set of statements of the form instr(g,g'). These statements capture the fact that the achievement of goal g is instrumental for the achievement of goal g'.
- A set of statements conflict(g,g') that explicitly denote conflicting goals. Authors, present ways to attack a goal by attacking beliefs subgoals or supergoals related to a particular goal. Finally they introduce a set of locutions that agents can use in dialogues. For example REQ-PURPOSE(g) allows an agent to ask for what supergoal the goal g is thought to be instrumental. This must be followed by an assertion involving instr(g,g') statement. REQ-JUST(g) allows an agent to ask other agents for the justification of a goal proposed by another agent. This must be followed by an assertion involving a justify(B,g).

Patricia Anthony et al. [4] present an *approach,* which is based on *heuristic-based* techniques for negotiation. They propose the construction of a software agent, which represents his owner and is able to search online auctions, negotiate with sellers and make purchases in an autonomous fashion. The bidding agent is given a termination threshold, the consumer's reservation price (consumer's personal evaluation of the product), the current bid of each individual auction and a set of tactics and strategies. The agent not only decides in which auction to participate but also what bid to offer. The restrictions stem from the strategies and from the time it has to buy the product that is the termination threshold. There are four factors which the agents consider in

order to calculate the current maximum bid at any given time t: (a) the remaining time (b) the number of remaining auctions (c) the level of desperateness and (d) the level of bargaining desire. All these are called the bidding constraints. The agent uses a set of polynomial functions, called tactics, to calculate the bid value based on a single constraint. The individual constraints are then combined to compose the agents overall position, that is, his strategy. Any tactic is allocated a weight that indicates its relative importance to the maximum bid estimation. Lastly they define the conditions for the choice of potential and target auction depending on auction category.

AuctionBot [24] is an auction management *system* supporting the creation, location and enactment of different kinds of auctions. Users can manually interact with the system through an HTML-based interface, or alternatively they can develop their own arbitrarily complex bidding agents and connect them to the auction manager through a TCP/IP-level API. This API is generic enough to deal with several kinds of auctions (e.g. English, Dutch, double, etc.) through a common set of primitives. The task of negotiation strategy is left to the user. Users must develop their own agents form the scratch every time they want to participate in a negotiation. There are no predefined strategies for users, so we cannot classify AuctionBot and relate it with a particular negotiation technique.

ContractBot [17] is an automatic contract negotiation *system* that integrates the three phases of e-contracting: discovery, negotiation and execution. A logic – based knowledge representation scheme, Courteous Logic Programs (CLP) is used to represent contracts and rules in general. A basic concept is that of contract template. It is a declarative description of all possible outcomes with additional rules, which influence the structure of negotiation. Contract template has two parts, the proto-contract and the negotiation level rules. The proto-contract refers to conditions of the deal such as the delivery options, payment options, warranties etc. It is the part of the contract that remains unchanged. The negotiation level rules answer questions like what is to be negotiated and how. In other words, they influence the structure of negotiation. Transactions in the auctions generate additional rules regarding buyers, sellers, prices, and quantities etc., which along with the proto-contract form the final contract. The system integrates with Michigan AuctionBot, which is the auction server.

Su et al. [26], present a web-based negotiation *system* for e-commerce, which uses heuristic *techniques*. They introduce an object-oriented content specification language,

which is based on active object model (AOM). The language and its accompanying GUI tools are used both by sellers to advertise their products and by buyers to express their preferences. These are stored in a persistent storage. The same language is also used by clients to define proposals and counter-proposals, which are wrapped in XML. A constraint satisfaction processing component is used for the evaluation of proposals and counter-proposals. Authors adopt a declarative approach and each negotiation strategy is expressed by means of event-trigger rules (ETRs'). The negotiation protocol is an FSM of allowed sequences of actions.

Tsang and Gosling [23] define the simple constrained bargaining game where one buyer interacts with one seller. They adopt a heuristic *approach* and try to find an optimal strategy, by comparing its profit against those by other strategies. The general rules are the following: The seller is constrained by a cost and the number of time units within which he must sell the product, time to sell (tts). The buyer is constrained by his utility and the number of time units he has to buy the product, time to buy (ttb). None of the participants have information about the other's cost, utility and time thresholds. The players make alternative bids with the seller to bid first. Each player bids exactly once per time unit. When both players bid for the same price, a sale is agreed. If a sale cannot be agreed before a player runs out of time, the negotiation terminates. No one has information about the others player's past behavior or performance.

Grosof et al. [12] extend their work described in [17] and incorporate process knowledge descriptions whose ontologies are represented in DAML+OIL. They give a conceptual approach to specifying LP/RuleML rules on top of DL/DAML+OIL language.

Rosencheim and Zlotkin [19] present rules of encounter for state-oriented domains. Their *approach* is based on game theoretic *techniques* for negotiation. In such domains, the goal of the agent is to move the world from an initial state to one or more desired states. There is interaction with other agents and limited resources. There is the possibility of cooperation, coordination, compromise and conflict. Any goal is described from the set of states that satisfy it. There are primitive operations that agents alone can do. When these operations are combined into a coherent sequence of actions specifying what both agents are to do, we speak about a joint plan. A joint plan transforms the world in a state that may or may not satisfy both agents. When agents carry out a joint plan, each agent plays some role. Their theory assumes that there is

some way of assessing the cost of each role. This measure of cost is essential to how an agent evaluates a joint plan. Among all joint plans, he will prefer the one with the higher utility. They use terms such as individual rational and Pareto optimal and tools from game theory to model agents' decision-making model.

Sierra et al [21] propose a formal *approach* for argumentation-based negotiation. They assume that a general and shared social relation is defined between agents. This relation can be modeled as a binary function over a set of social roles. The authors assume that participants exchange arguments in a common communication language CL defined over a set of argument particles, whose propositional content is expressed in a shared logical language L. CL accounts for the set of illocutionary particles, necessary to model the set of illocutionary acts. The acts can be divided in two sets Inego corresponds to negotiation particles such as offer, request, accept etc. and Ipers, corresponds to persuasive particles such as appeal, threaten, reward.

e-mediator [20] is an auction management server. It is a *system*, which uses a game theoretic and constrains satisfaction *techniques* for negotiation. It supports combinatorial auctions, in which a bidder may place bids on combination of items and may issue simultaneous bids for many combinations. It consists of three basic components. (a) *eAuctionHouse*, the configurable auction server, includes a variety of generalized combinatorial auctions and exchanges, pricing schemes, bidding languages, mobile agents, and user support for choosing an auction type. It introduces two new logical bidding languages for combinatorial markets: the XOR bidding language and the OR-of-XORs bidding language. Unlike the traditional OR bidding language, they are fully expressive. They therefore enable the use of the Clarke-Groves pricing mechanism for motivating the bidders to bid truthfully. *eAuctionHouse* also supports supply/demand curve bidding for single-item multi-unit markets. (b) *eCommitter*, the leveled commitment contract optimizer, determines the optimal contract price and decommitting penalties for a variety of leveled commitment contracting mechanisms, taking into account that rational agents will decommit strategically. It also determines the optimal decommitting strategies for any given leveled commitment contract. (c) *eExchangeHouse*, the safe exchange planner, enables unforced anonymous exchanges by dividing the exchange into chunks and sequencing those chunks to be delivered safely in alternation between the buyer and the seller. Each component can be used separately, but they can also be used together. For example, if a deal is reached using *eAuctionHouse* or *eCommitter*, then

eExchangeHouse could be used to carry out the deal. As another example, the contract determined using *eCommitter* could be auctioned using *eAuctionHouse*.

Travel Agent Game in Agentcities (TAGA) [39] is a framework that extends and enhances the Trading Agent Competition (TAC) scenario to work in Agentcities, an open multi-agent environment, based on FIPA compliant platforms. TAGA uses the semantic web languages and tools (RDF - OWL) to: (a) specify the underlying common ontologies, (b) as a content language within the FIPA ACL messages (c) as the base for agent knowledge bases via XSB-based reasoning tools (d) to describe and reason about services. TAGA extends FIPA protocols to support different types of auctions. The travel market which TAGA simulates includes service registries, service brokerage, wholesalers, peer-to-peer transactions, bilateral negotiation, etc. This provides a rich test bed for experimenting with agents and web services as well as an interesting scenario to test and challenge agent technology. TAGA operates as a continuous open game and anyone can participate and use his own agent strategy to compete with the others.

2.5 Brokering-Matchmaking

The second dimension of our research is the B*rokering* stage of e-commerce procedure. There is no consensus on the definition of terms brokering and matchmaking along with what functions they involve. However, most researchers ascribe almost identical characteristics and functionality to these terms. Thus, according to [31] brokering or matchmaking is the process by which parties that are interested in having exchange of economic value are put in contact with potential counterparts. According to [28], brokering or matchmaking is defined as a process that requires a repository host to take a query or advertisement's input and to return all advertisements that may potentially satisfy the requirements specified in the input query or advertisement. [29] Defines that broker is someone who offers a matchmaking service to potential service providers and service requesters. From now on we will use the term brokering, referring to both terms.

The main dimensions of brokering are A*dvertisements, Preferences & Queries and Brokering Engines.* An advertisement models the offers for a service of a potential service provider and encloses service's features, constraints and special characteristics. It defines a space of possible realizations of a service. The query, which may also be a request advertisement, expresses constraints over aspects of advertised services that the

submitter is interested in. It is a way for the requester to filter out existing advertisements that are not important to him. Lastly, Brokering engine is a mechanism, which employs a specific technique and does the act of brokering, which is the matching of the preferences of the service requesters and the advertisements of the service providers. A set of advertisements (subset of the total of advertisements) is returned to the service requester.

2.5.1 Brokering Techniques

Contrary to the field of negotiation, we did not find a complete classification of the brokering techniques. After a review of the related literature we shaped a classification scheme but it is possible that other techniques may exist.

In [1], three techniques for brokering are identified. They are called *Feature-based* filtering, *Collaborative Filtering* and *Constrained-based Filtering*.

- Feature-based filtering involves selecting products based on feature keywords. For example, suppose a customer wants to buy a Sony notebook computer through Amazon. His agent selects the "Computers" category first, and then indicates "Sony" in the brands field, and the notebook computers with these features are returned.
- Collaborative filtering involves giving an agent personalized recommendations based on the similarities between different users' preference profiles. Here, the product rating of shopper A is first compared with that of all the other shoppers in the system. Then, the "nearest neighbor" of A (i.e., the shopper whose profile is closest to that of A) is identified. Since shoppers with similar tastes and preferences are likely to buy similar products, the profile of the identified shopper is used to pass recommendations onto A's agent.
- Constraint-based filtering involves an agent specifying constraints (e.g., the price range and date limit) to narrow down the products. In this way, customers' agents are guided through a large feature space of the product. In the end, a list of the desired products that satisfy the user's constraints is returned. Some e-commerce systems use more than one kind of filtering technique (since sometimes users do not know exactly the constraints of the goods they are looking for in advance).

In the last few years declarative approaches start to emerge. In addition many of these approaches start to adopt semantic web technologies. There are many variations for brokering which are based on declarative approaches:

- A common technique uses a logic programming language for the modeling of both the advertisements and the queries. In this case, a rule-based inference engine is used for the brokering procedure.
- A more recent technique makes use of the data model of RDF for the description of both the advertisements and the requests. It uses a matching algorithm as the brokering engine; thereby matching of advertisements is reduced to matching of RDF graphs.
- Both advertisements and requests can also be expressed in a description logic-based web ontology language, such as OWL or DAML+OIL. In this case, advertisements and requests are converted to Aboxes and Tboxes and afterwards a description logic reasoner is used.
- An interesting technique could be the use of RDF for expressing advertisements and the use for a query language like those described in [51] for expressing queries of service requesters.
- Finally, a hybrid approach is possible. The idea is that advertisements are expressed in a description logic based web ontology language or a data model such as RDF, but the queries and preferences of users are expressed through a logic programming language. In this case, the brokering engine must be able to convert description logic to declarative logic, in order to perform the reasoning process.

2.5.2 Fundamental Requirements for a Brokering System

The minimal functionalities that a brokering service provides are the features of a *Messaging Middleware*, *Advertising* a service, and *Browsing* or *Querying* a repository of advertised services [31]. More specifically:

- A messaging middleware must exist that allows message exchange among service requester service provider and broker.

- There is a need for a language to express advertisements. This language must be flexible and highly expressive, support to express semi-structured data, types, subsumption and constraints.
- The existence of a language for expressing queries is indispensable for a brokering system. If queries are in the form of request advertisements, the characteristics of a language for advertisement also apply here. In addition the ability for the expression of complex queries is critical.

2.5.3 Selected Work in Brokering

Net Perception and CDNOW are two brokering *systems*, which use a collaborative *technique*. In Net Perceptions [79], users are recommended the documents that their "knowledge neighbors" find valuable. In CDNOW [80], users are notified about the CDs or movies that are popular with other users with similar preferences. eBay, [78] guides a user agent to select the products by narrowing down the range of the possibilities based on the constraints the user gives (e.g. price range, item location, and so on). In the end, a list of the desired products that satisfy the user's constraints is returned. It uses a combination of constraint and feature-based techniques.

InfoSleuth [29] is an agent-based information discovery and retrieval *system*, which performs a logic programming brokering. It adopts "broker agents" to perform the syntactic and semantic matchmaking. The broker agent matches agents that require services with other agents that can provide those services. By maintaining a repository, which contains up-to-date information about the operational agents and their services, the broker enables the querying agent to locate all available agents, which provide appropriate services. Syntactic brokering is the process of matching requests to agents on the basis of the syntax of the incoming messages which wrap the requests; semantic brokering is the process of matching requests to agents on the basis of the requested agent capabilities or services, with the agent capabilities and services being described in a common shared ontology of attributes and constraints. This single domain-specific ontology is a shared vocabulary that all agents can use to specify advertisements and requests to the broker. In InfoSleuth, the service capability information that is regarded as the advertisement is written in LDL++, a logical deduction language. Agents use a set of LDL++ deductive rules to support inferences about whether an expression of requirements matches a set of advertised capabilities.

Trastour et al. [31] propose an *approach,* based on a matching of RDF graphs technique. After they enumerate the minimal requirements for a brokering service, they examine the currently available standards like UDDI and ebXML and find they fall short of the requirements for a brokering service. They propose the use for RDF and describe an advertisement as an RDF graph. As they state, advertisements and queries have much in common so, queries at their approach are request advertisement. They implement a matching algorithm based on the visitor pattern, using the Jena RDF API. Their approach does not use a persistent repository for the storage of advertisements.

In similar lines, Li and Horrocks [28] propose an *approach* by exploiting the use of DAML-OIL, description logic based web ontology language. Another distinction is that there is no matching of DAML+OIL graphs. On the contrary, they choose to use RACER description logic reasoner to compute semantic matches between service advertisements and service requests. DAML+OIL is transformed to Aboxes and Tboxes and are fed into the reasoning engine. At the beginning of the matchmaking process, the HostAgent initializes RACER with a service ontology, which the RACER system will use to compute the subsumption relations between advertisements and requests throughout the whole matchmaking process. When it receives an advertisement, the HostAgent assigns it a unique ID and stores it in the repository. It then sends the advertisement's ServiceDescription to the RACER system to be added to the subsumption hierarchy. When it receives a request, the HostAgent uses the RACER system to compute all the match degrees between the request and each advertisement in the repository. Matching advertisements are returned to the seeker agent, along with their IDs and match degrees (Exact, PlugIn, etc.). For efficiency reasons, match results for a persistent request are maintained until the request expires. Similar is the approach adopted by Massimo Paolucci et al.[30].

Chen et al. [27] propose a *hybrid* technique. The iAgent stores knowledge in the inference layer and makes inference by it, so iAgent can answer the queries of users or other agents. As they stress, a typical knowledge base makes inference according to the *rules* and *facts*, but they choose not to predefine rules and facts in the inference layer. Such design makes the inference layer as portable as possible, so different iAgents in different domains can use the same inference layer. The facts are extracted from semantic markup documents which are written in DAML+OIL. A fact translator converts all the DAML+OIL documents into Prolog formats. iAgent chooses a Prolog (Horn-logic based) engine, SWI-Prolog, as its inference engine.

GraniteNights [32] is a multi-agent visit scheduler *system*, which utilizes semantic web technologies. Agents are organized according to a set of predefined roles. The basic functional agents of the system are: (a) information agents, which are wrappers for RDF resources (either static or dynamically generated from existing WWW sources). These agents also have a query module and receive RDF queries. (b) profile agents, manage user data such as id password and preferences (c) scheduler agent is a constraint-solver agent which maps RDF that have been returned by information agents to finite domain constraints, combines them with user preferences and produces valid instantiated schedules. (d) Evening agent receives user's queries and preferences and generates a solution. Finally, there is a user-interface agent. There is a module, which generates RDF messages from simple web forms filled by the users. The application exploits JADE platform, SICstus prolog with jasper module as well as Blue JADE.

3. On the Use of Defeasible Logic - Proposed Solution

In this section we justify our choice of defeasible logic, as the basic knowledge representation language for modeling strategies in automated negotiation cases and preferences of users in semantic brokering cases.

Subsequently we provide an abstract architecture both for the negotiating agent and the brokering system we propose. A more detailed description of these architectures is presented in section 4.

Finally, we show how our approach seems to be better or at least complementary to other approaches and model indicative examples in both fields of negotiation and brokering, using defeasible logic. Particularly we model: (a) the strategy of an agent which participates in multiple auctions, (b) the strategy of an agent for an english auction, (c) the requirements of an agent which seeks a brokering service, (d) a strategy of a seller agent and (e) the strategy of two agents, which participate in a multi-attribute 1-1 negotiation.

3.1 Eligibility of Defeasible Theory for Negotiation and Brokering

Applying a negotiation strategy in a particular context is an intensive decision-making process. While most aspects of negotiation strategies could be fully captured in classical logic programming (which has a formal semantics and has proven to be a powerful tool for building decision-making systems), this would put a burden on the developers of strategies, since logic programming is a generic paradigm and offers nothing specific to strategy specification (such as argumentation, defeasibility, hypothetical reasoning, preferences, etc.). Accordingly, we propose to use a logic programming language based on non-monotonic reasoning. Among the many members of the family of non-monotonic logics, we choose defeasible logic [53] for the following reasons:

- A negotiation can be thought of as a dialogue between parties concerning the resolution of a dispute. This suggests that argumentation based reasoning formalisms are suitable to characterise it. In [57], it was shown

that defeasible logic can be characterised by argumentation semantics, thus the formal semantics of defeasible logic is in line with the argumentative nature of negotiations.

- Given the close connection between derivations in Defeasible Logic and arguments, strategies expressed in Defeasible Logic are explainable.
- Defeasible logic is a sceptical formalism, meaning that it does not support contradictory conclusions. Instead it seeks to resolve conflicts. In cases where there is some support for concluding A but also support for concluding ¬A, the logic does not conclude either of them (thus the name "sceptical"). If the support for A has priority over the support for ¬A then A would be concluded. We believe that non-sceptical reasoning is inappropriate for modelling decision-making processes such as negotiations, since it is quite useless to deduce both that a decision should be taken, and that it should not be taken.
- Defeasible logic integrates the concept of priorities between rules, thereby supporting a direct way of modelling preferences, without having to attach a metric to them, as it is the case of approaches based on utility functions [16].
- Regarding strategy specification, most of the current systems adopt a quantitative approach based on utility functions. Very often, it is not easy to find the right utility functions for a given set of negotiation issues, especially in situations where one needs to express preferences without attaching a metric to them. Moreover, utility functions are mostly used to determine preferences that can otherwise be expressed in a more comprehensible and suitable way through defeasible rules and priorities among these rules. For this reason, we believe that defeasible logic is more suitable than, or at least complementary to, strategy specification approaches purely based on utility functions.

Before choosing one or several languages for the specification of requests and preferences of users, it is important to establish a set of criteria that such languages need to satisfy. The criteria presented below are inspired from those formulated by [64]

in the context of techniques for information modeling. They encompass several well-known principles of language design.

Firstly, a language for specifying requirements and preferences needs to be *formal*, in the sense that its syntax and its semantics should be precisely defined. Secondly, the language should be *conceptual*. This, following the well-known *Conceptualization Principle* of [65], effectively means that it should allow its users to focus only and exclusively on aspects related to requirements, without having to deal with any aspects related to their realization or implementation. Thirdly, in order to ease the interpretation of strategies and to facilitate their documentation, the language should be *comprehensible*. Comprehensibility can be achieved by offering a graphical representation, by ensuring that the formal and intuitive meanings are as much in line as possible, and by offering structuring mechanisms (e.g. decomposition). These structuring mechanisms often lead to *modularity*, which in our setting means that a slight modification to a strategy should concern only a specific part of its specification. As we are interested in the automation of the brokering process, the requirements description language should be *executable.* Finally, the language that we aim should be sufficiently *expressive*, that is, it should be able to precisely capture a wide spectrum of requirements.

We have chosen *defeasible logic* to represent requesters' requirements and preferences because it satisfies the above criteria. In particular,

- It is a formal language with well-understood meaning ([53] presents a proof theory, [61] its model semantics, and [57] its argumentation semantics), thus it is also predictable and explainable.
- It is expressive, as demonstrated by the use of rules in various areas of information technology. In fact, among the logical systems, it is rule-based systems that have been best integrated in mainstream IT.
- Finally, it is suitable for expressing requirements and preferences in our setting. This is so because it supports the natural representation of important features:
    - Rules with exceptions are a useful feature in our problem. For example, a general rule may specify acceptable offerings, while more specific rules may describe cases in which the general rule should not

apply and the offering should not be accepted. We will elaborate on this point in the next section when we consider a concrete example.

- Priorities are an integral part of defeasible logic, and are useful for expressing user preferences for selecting the most appropriate offerings from the set of the acceptable offerings.

Lastly something, which applies in both cases, is the following:

- Defeasible logic has a linear complexity, and existing implementations [62] are able to deal with non-trivial theories consisting of over 100,000 rules [60], offering thus an executable and scalable system.

3.2 Proposed Solution

3.2.1 Abstract Negotiating Agent Architecture

The two basic roles that we identify in the negotiation case are the *Buyer* and the *Seller*. There is now yellow pages service (or Matchmaker) for the negotiation and we make the assumption that buyer that starts the negotiation "*knows*" the address of the seller. The technical solution we provide is based on the following key ideas:

- The buyer and the seller are represented by software agents.
- The negotiation strategy of the agent is expressed using defeasible logic rules.
- The knowledge base of the agent consists of a set of facts and a number of defeasible and strict rules.
- Both buyer and seller use a hard-coded negotiation protocol.
- When the agent receives an offer from somebody, he stores the new facts to the knowledge base (K.B.). Consequently, the control module activates the inference engine, which in turn updates the knowledge base with the inference result, according to the strategy rules. Finally the control module retrieves the result and posts it to the communication module of the agent.

The abstract architecture of the negotiating agent is depicted in Fig.11. A concrete architecture is discussed in section 4.

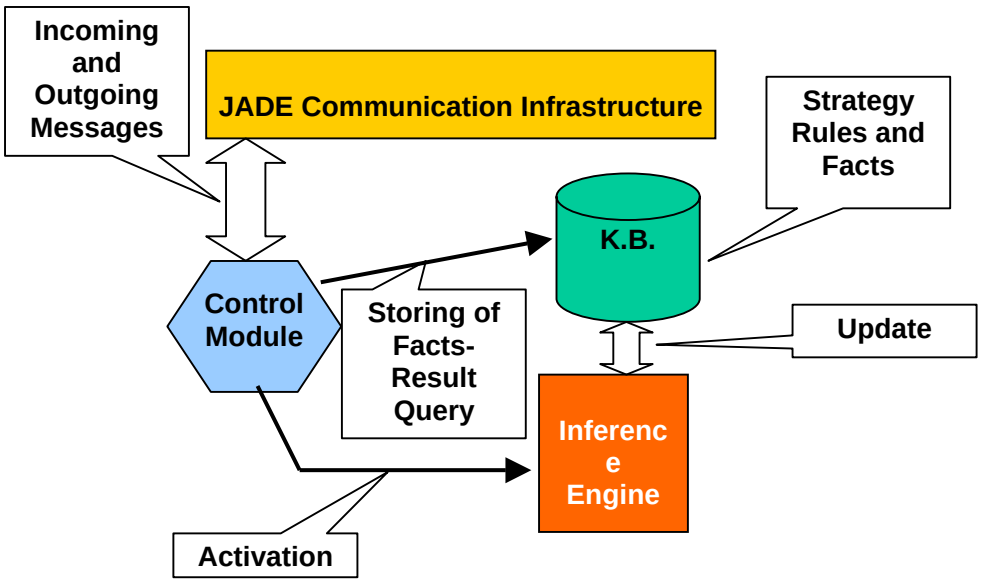


Figure 11. Abstract Negotiating Agent Architecture

3.2.2 Abstract Brokering System Architecture

The three basic roles that we identify in our brokering system are the *Service Requester* (or the Buyer), the *Service Provider* (or Seller), and the *Broker*. Another agent, called Directory Facilitator (D.F.) plays a secondary role and is the yellow pages service (or Matchmaker), which agents use to find each other and register what protocols they use, what ontologies they use, etc.. The technical solution we provide is based on the following key ideas:

- Service requesters, service providers and brokers are represented by software agents.
- The requirements of the service requester are represented in defeasible logic, using rules and priorities. These requirements include both indispensable requirements that must be met for a service to be acceptable (for example, air-conditioning is required), and soft requirements (preferences) that can be used to select among the potentially acceptable offerings. These requirements are communicated to the broker agent by the requester agent. This communication initiates a brokering activity.

- The offerings are represented in a certain semi-structured format using the Semantic Web standard language RDF for describing Web resources. The provider agents communicate the offerings to the broker agent.
- The terminology shared by providers, requesters and brokers is organized in an ontology using the Semantic Web standard of RDF Schema.
- The broker is also a software agent and has special knowledge both for the declarative language and the advertisement format. It also has the ability to perform semantic checks to the information it receives.
- When the broker receives a request it matches the request to the offerings by running the request specification against the available offerings, making use of information provided by the shared ontology, as required. Then the requester's preferences are applied to select the most suitable offering(s) which are then presented to the requester.
- For the persistence storage of advertisements, an RDF repository is used.

The abstract architecture of our brokering system is depicted in Fig. 12. A concrete architecture can be found in section 4.

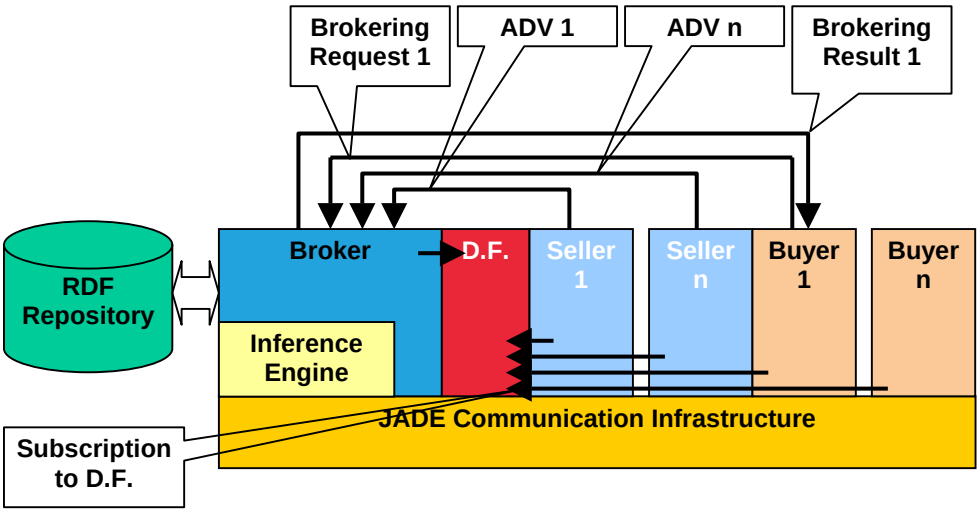


Figure 12. Abstract Brokering System Architecture

3.3 Comparing our Approach to the Others

3.3.1 Brokering

In this subsection, we compare the related work in brokering to ours, based on the fundamental requirements for a brokering system that we have already posed.

When it comes to feature-based, collaborative and constrained-based filtering implementations, they fall short of the support of semantic web technologies although they have been successfully used for many years. The same happens with systems, which express both advertisements and queries in a logic programming language. In our implementation for brokering, we adopt semantic web technologies as we express advertisements in RDF and use an RDF repository for their storing.

Approaches, which use RDF for the expression of both the advertisements and the requests, seem to fulfill most of the requirements of a brokering service. Particularly they are flexible solutions that support semi-structured data, types and subsumption. One problem with these approaches is their inability to express complex queries. In addition RDF intrinsically has one more restriction. It does not support constraints, except for type and equality ones. For example it cannot restrict the range of a property, for a subset of the initial domain. Consider for example that someone advertises that he is

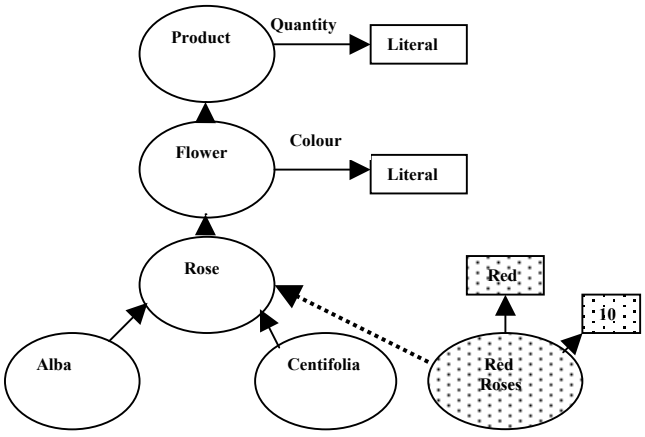


Figure 13. RDF's Limitation in Constraints Expression

interested in purchasing 10 Kg of Red Roses. It is also known that Alba is roses that are always white. If there is an advertisement for 10 Kg of Alba, it will be in the result set, something undesirable. In our brokering implementation, we also use RDF for the

advertisements but we use defeasible logic rules for the requests and user's preferences. The mentioned problem can be overcome in the following way:

r_1: Rose(X), Quantity(X,Y),Y=10 => accept(X)
r_2: Rose(X), Quantity(X,Y),Y=10,X=Alba => ¬accept(X)
$r_2 > r_1$

One way to overcome the above limitations of RDF is to use a description logic – based web ontology languages such as DAML + OIL and OWL for the expression of the advertisements and the queries. The problem in this case is the high computational complexity. In contrary, as we have already said it has been proved that defeasible logic has linear computational complexity [60].

Another approach is the use of RDF for expressing advertisements and the use of a query language such as RQL for expressing queries. Although RQL seems to have high expressive power compared to other query languages [51], it is very difficult, if not impossible, to capture user's soft requirements. When it comes to requirements, we separate indispensable requirements that must be met for a service to be acceptable (for example, air-conditioning is required), and soft requirements (preferences) that can be used to select among the potentially acceptable offerings (central heating is most important for me than air-condition). Consider the following scenario: Somebody is ready to buy a house with air-condition and price lower than 10000 € or a house with central heating and price lower than 15000 €. However, the second condition has priority for him over the first. It is very easy for someone to capture all kinds of requirements in our implementation.

r_1: hasAC(X), Price(X,Y),Y<1000 => accept(X)
r_2: hasCentralHeating(X), Price(X,Y),Y<1500 => accept(X)
$r_2 > r_1$

Finally GraniteNight's approach uses a constraint solver, something which is orthogonal to our proposed solution.

3.3.2 Negotiation

In this section we compare our negotiating agent architecture to other solutions and examine whether the realization of negotiation strategies, classified in all the negotiation techniques is possible.

Kasbah an older system and e-Mediator, which is more recent, are well-known negotiation systems. The former provides an easy GUI and enables user to pick among

some predefined strategies for bid increase. The latter uses game theoretic and constrains satisfaction *techniques* for negotiation. It supports combinatorial auctions, in which a bidder may place bids on combination of items and may issue simultaneous bids for many combinations. However, none of them allows a user to define his own strategies, leading to an inflexible solution. Our implemented negotiating agent architecture solves this problem as the user every time loads dynamically the rules of the strategy that correspond to user's exact preferences.

Another characteristic system for negotiation is Auction Bot that is the base for the former Trading Agent Competition (TAC), which today has evolved to the Travel Agent Game in Agentcities (TAGA). All the agents that were designed for Auction Bot and most of the agents, which participate in TAGA, employ a hard-coded implementation of the agent (concerning its reasoning module), something that is time-consuming. In addition it is very difficult to change the behavior of the agent without having to rebuild it form the scratch. In our solution, a few modifications to a subset of the rules or the loading of new rules can completely alter agent's behavior without great effort.

There is a plethora of work when it comes to negotiation techniques. Defeasible logic is able to represent strategies of all these categories. In the case of argumentation-based approaches they can be easily modeled through defeasible predicates, shared by the participants of the negotiation. In the case of game theoretic approaches, utility functions can be easily attached on the defeasible rules. Finally, there is no problem with heuristic techniques. This is adequately proved by the examples of this thesis, which are in majority heuristic approaches.

3.4 Modeling Negotiation and Brokering Examples in Defeasible Logic

3.4.1 An Agent which Participates in Multiple Auctions

Motivation

This case study concerns the work of Patricia Anthony et al [4]. According to the authors, there is an increase in online auctions during the last few years. So, customers who seek to make beneficial deals have to monitor many auctions simultaneously select the most appropriate and make a bid in order to buy the desired item in line with

their strategy. As these auctions have different start and termination times, the above processes tend to be time consuming.

There are two services that can be viewed as a solution. Some auction houses offer agents, which are supplied with customer's starting and maximum bid, and increase the starting bid progressively. There are many shortcomings to this approach. Firstly, the number of auctions the customer can participate reduces. In addition, if a customer finds another auction that seems more interesting to him, he can't offer a bid, as the robot may has already placed a bid and the customer risks to buy the product twice. Another service is that of auction search service, which monitors multiple on-line auctions, but lives the bid decision to the customer. So the latter is bound to be online a lot of time again. In addition the customer may overestimate the value of a product.

The algorithm

In such a context the authors propose the construction of a software agent that represents his owner and is able to search online auctions, negotiate with sellers and make purchases in an autonomous fashion. The bidding agent is given a termination threshold, the consumer's reservation price (consumer's personal evaluation of the product), the current bid of each individual auction and lastly a set of tactics and strategies (Fig.14 borrowed from the work of [4]).

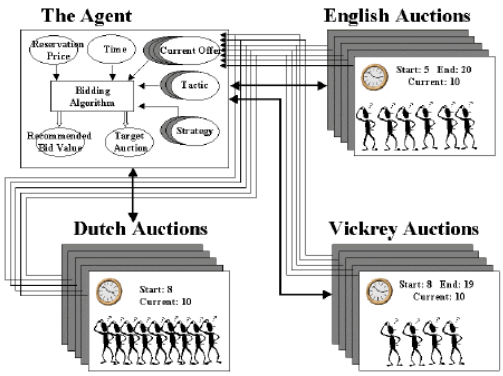


Figure 14. Strategy of Agent which Participates in Multiple Auctions

The agent not only decides in which auction to participate but also what bid to offer. The restrictions stem from the strategies and from the time it has to buy the product, which is the termination threshold. The algorithm the agent employs is the following:

```
While (t<tmax) and (Item_NA=true)
 Build active auction list ;
 Calculate current maximum bid
using agent's strategy ;
 Select potential auctions to bid in
from active auctions list;
Select target auction as one that
maximizes customer's utility;
Bid in target auction using current
maximum bid as reservation price;
End While
```

tmax is the termination threshold and Item_NA is a Boolean indicator of whether the item has been bought. The active auction list is all the auctions that have started but not yet reached their end time. Current maximum bid is the amount the agent is allowed to bid at the current time, in line with its strategy.

Calculating current maximum bid and selecting potential auctions

There are four factors which the agents consider in order to calculate the current maximum bid at any given time t: (a) the remaining time, (b) the number of remaining auctions, (c) the level of desperateness and (d) the level of bargaining desire. All these are called the bidding constraints. The agent uses a set of polynomial functions to calculate the bid value based on a single constraint. These functions are called the tactics. The individual constraints are then combined to compose the agent's overall position that is its strategy. Any tactic is allocated a weight that indicates its relative importance to the maximum bid estimation. The current maximum bid at a time t is the following:

$$M(t) = \sum_{j \in C} w_j(t) f_j(t),$$

where $w_j(t)$ is the weight on constraint j at time t, $\sum_{j \in C} w_j(t) = 1$ and $0 \le w_j(t) \le 1$

$f_j(t)$ is the bid value according the constraint j. For further information about the four tactics refer to [4].

Now let's see how the agent selects potential auctions. There are some basic rules. The agent selects new auctions if and only if it is not holding the highest bid in an English auction and it has not placed a bid in a Dutch or Vickrey auction.

Only English auctions that are close to their end time are chosen, to maximize agent's chances of winning. The potential Dutch auctions in which the agent may bid are those with current bids that are less than the current maximum bid. The potential Vickrey auctions are those that are terminating at the current time and the agent offers his current maximum bid value. If there are many auctions of each type, the agent calculates the expected utility for each auction. The target auction is that with the highest expected utility, where the agent offers his current maximum bid.

We use two correlated defeasible theories. One for the general control of the algorithm, and one for the auction filtering. We omit the auction selection stage. During the auction filtering the active auction list is constructed while during the auction selection the agent selects one auction to bid in. The predicates we use are the following:

- currentBid(A_id, Vt) : The current offered bid Vt (dominant) at auction A_id.
- currMaxBid(Mt): The current maximum bid of the agent.
- auctionType(A_id, Type): The type (english, vickrey, dutch) of auction A_id.
- bidIncr(A_id, incr): The bid increment for an auction A_id.
- closeTime(A_id, tend): The termination time of auction A_id.
- selectable(A_id): Determines whether an auction belongs to the active auction list.
- bidPlaced(A_id, Mt): The amount of bid Mt that agent offered at auction A_id.
- buyThreshold(tmax): The available time for the agent to buy the product.
- Time(t) the time of the negotiation.

Control Module

r_1: $\Rightarrow$ monitor

r_2: time(t), buyThreshold(tmax), t $\geq$ tmax $\Rightarrow \neg$ monitor

r_3: item_bought $\Rightarrow \neg$ monitor

$r_2 > r_1$

$r_3 > r_1$

Rule r_1 signals the start of the monitor procedure, while r_2 and r_3 terminate it if either the threshold of the agent expired, or he bought the product.

Auction Filtering

r_1: currentBid(A_id,Vt),bidPlaced(A_id,Mt),auctionType(A_id,Type),Type=english,Vt=Mt ⇒ wait

r_2: bidPlaced(A_id,Mt),auctionType(A_id,Type),Type=vickrey ⇒ wait

r_3: bidPlaced(A_id,Mt),auctionType(A_id,Type),Type=dutch ⇒ wait

r_4: currentBid(A_id,Vt),auctionType(A_id,Type),bidIncr(A_id,Incr),closeTime(A_id,Tend),currMaxBid(Mt),time(T),T<Tend,Tend-T=1,Type=english,Vt+Incr≤Mt ⇒ selectable(A_id)

r_5: currentBid(A_id,Vt),auctionType(A_id,Type),time(T),closeTime(A_id,Tend),currMaxBid(Mt),Type=dutch,Vt≤Mt,T<Tend ⇒ selectable(A_id)

r_6: auctionType(A_id,Type),closeTime(A_id,Tend),time(T),type=vickrey,Tend-T=1,T<Tend ⇒ selectable(A_id)

$r_1, r_2, r_3 > r_4, r_5, r_6$

Rule r_1 states that if the agent has placed a bid in an English auction and it is the highest, he must wait until he is overridden, to avoid buying the product twice. According to r_2 and r_3 if the agent has placed a bid in a Vickrey or Dutch auction he must also wait until he is overridden. It is obvious by r_4 that an English auction is selectable only, if it is about to finish and the current bid value when added to bid increment is less than the current maximum bid. Rule r_5 states that potential Dutch auctions are those with current bid less than the current maximum bid. Lastly rule r_6 states that only those Vickrey auctions which terminate at the current time are selectable.

Table 1. The Set of Monitored Auctions

A_id	currentBid	currMaxBid		bidIncr	closeTime	auctionType	Time
e1	50	70	80	10	100	english	80
e2	60	70	80	15	81	english	80
d1	200	70	80	-	50	dutch	80
v1	-	70	80	-	81	vickrey	80
d2	70	70	80	-	81	dutch	80
v2	-	70	80	-	40	vickrey	80
e3	40	70	80	5	70	english	80

Table 1 presents a set of auctions along with their characteristics. We firstly consider that the agent can offer at most 70 money units (currMaxBid = 70) and afterwards we examine the case that he can offer 80 money units (currMaxBid = 80). Based on the auction-filtering algorithm and when currMaxBid = 70 we see that: (a) Auction e1 is not selectable because it is not about to finish (rule r_4). (b) Although auction e2 is about to finish, it is not selectable because the currentBid plus the bidIncr is greater than the amount the agent can offer, which is currMaxBid (rule r_4). (c) Auctions d1, v2 and e3 are not selectable because they have terminated. (d) Auctions v1 and d2 are selectable (rules r_5, r_6).

If currMaxBid is 80, auction e2 is added to the set of selectable auctions as now the condition $Vt+Incr \leq Mt$ of rule r_4 is not violated.

3.4.2 An English Auction Bidding Agent

The English auction is perhaps one of the most popular one-to-many negotiation mechanisms. In its simplest form, it serves to select a buyer for an item and to establish its price (multi-party single-issue negotiation). There are many variants of the English auction. The variant that is currently in use within the biggest online auction houses (e.g. eBay [78]) may be roughly described as follows. The seller starts by setting a reservation price, which may or may not be announced to the bidders. He also sets a timing constraint, which may be either expressed as a firm deadline, as a maximum duration between two successive bids, or as both. Potential buyers then issue increasingly higher bids. The increment between one bid and the next is constrained to be greater than a given threshold. The auction stops when the timing constraint is violated, i.e. either the deadline is reached, or no bid is registered for longer than the established maximum duration. The last bidder then buys the item at the price of the last bid. If no bid is issued at or above the reservation price, the item is not sold.

To illustrate how a bidder's strategy is expressed using defeasible logic, we consider the following scenario. A user wishes to participate in the auction of an item. He doesn't know exactly how much the item is worth, but he thinks that its value lies somewhere within two bounds L and U. The user is keen not to over-value the item, so he decides to assume at the beginning of the auction that the item is worth L, and to eventually increase his valuation whenever one of the following two situations occurs: (a) at least 3 bids above his current valuation have been registered, or (b) somebody has bid more than 20% of his current valuation. As soon as one of these conditions is met, the user will raise his valuation by the minimum possible amount that allows him

to stay in the auction. However, he will never value the item above U. As usual in the case of English auctions, the user will start by bidding the reservation price (or the minimum possible bid if the reservation price is not known), and he will subsequently overbid the other participants' bids by the minimum increment, as long as the resulting bid is less than his current valuation. However, if the auction's deadline is too close, he will bid his current valuation instead of just overbidding by the minimum increment.

Formally, the parameters, status, and history of the auction, are modeled through the following predicates and constants:

- Constant min_increment denotes the minimum amount by which the bidders are allowed to overbid.
- Constant initial_bid denotes the minimum amount of the first acceptable bid.
- Predicate time_remaining(T) gives the time remaining before the end of the auction.
- Predicate highest_quote(N) provides the current highest bid.
- Predicate quotes_above(X,N) holds if N bids above amount X have been registered.

The last two predicates (highest quote and quotes above) provide aggregate views over the history of the negotiation process as seen by the agent. This history is captured by a predicate history(L), where L is a list of pairs <time, proposal>. In the case of an auction, the proposals are the price quotes received by the agent from the auction broker. The rules for deriving predicates highest_quote and quotes_above out of predicate price_quotes are all strict (i.e. not defeasible), and therefore we omit them in the sequel.

The parameters and decision rules of the user's strategy are modeled by the following constants and predicates:

- Constant time_threshold is the duration to the deadline, below which the user estimates that he should bid his valuation instead of just overbidding by the minimum increment.
- Constant significant_bidders is the number of bidders that should bid above the user's current valuation before he considers raising it.

- Constant significant_increment is the amount (expressed as a percentage), which another bidder should bid above the user's current valuation before he considers raising it (in the working example this is 0.2).
- Constant max_valuation is self-explanatory.
- Predicate submit_bid(X) states that a bid of amount X should be submitted.
- Predicate valuation(X) gives the current valuation while pre valuation(X) gives the valuation that was valid at the end of the previous activation of the reasoning module.

The reasoning module that we develop here can be easily adapted to capture strategies where the increment between one bid and the next is gradually increased as the deadline approaches.

- Predicate my_bid(X) gives the amount of the last accepted bid issued by the bidder. At the beginning of the auction my bid(0) holds.

The rules that model the strategy of the bidding agent are the following:

conflict:: valuation(X), valuation(Y + min_increment)

r_1:my_bid(X),highest_quote(Y),valuation(Z),X<Y,Y+min_increment<Z,time_remaining(T),T>time_threshold ⇒ submit_bid(Y+min_increment)

r_2:my_bid(X),highest_quote(Y),valuation(Z),X<Y, Y + min_increment<Z,time_remaining(T),T≤time_threshold ⇒ submit_bid(Z)

r_3: pre_valuation(X) ⇒ valuation(X)

r_4:pre_valuation(X),quotes_above(X,N),N≥significant_bidders, highest_quote(Y) ⇒ valuation(Y + min_increment)

r_5:pre_valuation(X),highest_quote(Y),Y>(1+ significant_increment) * X ⇒ valuation(Y + min_increment)

r_6: Y > max valuation → ¬ valuation(Y).

$r_4, r_5 > r_3$.

Rules r_1 and r_2 model the bidding strategy: Rule r_1 states that if there is enough time remaining and the agent's current bid is not the highest one, it should be increased by the minimum increment, provided that the current valuation allows so. Rule r_2 states that if the deadline is close and the bidder does not hold the item, a bid of the amount

of the current valuation should be submitted immediately. Rules r_3 through r_6 model the evolution of the valuation: Rule r_4 and r_5 model the two conditions under which the valuation should be raised, while rule r_6 is a defeater modeling the fact that the bidder is under no circumstances willing to value the item above a given amount. The use of this defeater provides a strong modularity to the defeasible program. If for instance the user wanted to modify the above strategy with a statement of the form raise the valuation if the reservation price has not been met and the highest bid is above my current valuation", then he just has to extend the above defeasible logic program with the following rule :

```
r7 :reservation_not_met,valuation(X),highest quote(Y),Y>X
⇒ valuation(Y + min increment)
     r7 > r3
```

without having to worry, whether the reservation price is greater than his maximum valuation or not.

3.4.3 A Brokering Scenario

In this subsection we present a brokering scenario. Bob, who holds a middle management position, looks for an appropriate hotel room for his business trip to Athens. He wishes to stay at a central hotel, or at least at a hotel close to public transport. And he requires the hotel to have air-conditioning, a gym, and generally speaking a business standard.

In accordance with tight budgeting rules of his company, Bob is willing to pay a modest price: 70 Euros per night. However, if the hotel is central he is willing to pay 80 Euros, and if the hotel has a pool he is willing to pay 90 Euros. If given the choice, he would go for a hotel with a central location, with the lower price being his secondary preference criterion.

Formalization of Requirements

We show how Bob's firm requirements are represented in defeasible logic. The predicate *acceptable(X)* is used to denote that a hotel is acceptable. The first rule says that, a priori, all hotels are acceptable.

```
conflict:: offer(X,70), offer(X,80), offer(X,90)
r1: ⇒ acceptable(X)
```

However, any hotel not satisfying one of the required features is unacceptable. The following rules describe exceptions to the first, more general rule. In fact, rules with

exceptions are a common, very useful representational mechanism of defeasible logic. Note that the exception rules are declared to be stronger than the general rule.

r_2: ¬central(X),¬publicTransport(X)⇒ ¬acceptable(X)

r_3: ¬gym(X)⇒ ¬acceptable(X)

r_4: ¬aircon(X) ⇒ ¬acceptable(X)

r_5: ¬business Standard(X) ⇒ ¬acceptable(X)

$r_2 > r_1$, $r_3 > r_1$, $r_4 > r_1$, $r_5 > r_1$

Next we must represent the price Bob is willing to pay at most. The predicate offer(X,Y) denotes that Bob is willing to pay at most Y Euros for hotel X.

r_6: ⇒ offer(X,70)

r_7: central(X) ⇒ offer(X,80)

r_8: pool(X) ⇒ offer(X,90)

$r_8 > r_7 > r_6$

A hotel is unacceptable if its price is higher than what Bob is willing to pay. This rule is also an exception to the general rule r_1.

r_9: price(X,Y), offer(X,Z), X>Z ⇒ ¬acceptable(X)

$r_9 > r_1$

Representation of Offered Hotels

The hotel offerings are represented as logical facts. For example, hotel h1 can be described by the facts:¬central(h1), aircon(h1), publicTransport(h1), category(h1,2), gym(h1), price(h1,50), ¬pool(h1). Table 2 shows seven offerings in table form. It is interesting to look at the category information, which does not appear in the rules describing Bob's requirements. This is natural since Bob, coming from a different country, does not know the meaning of the hotel ratings in Greece. It is an ontology of the tourism domain that would establish a link between the two. In our example, we assume that business standard is provided by Greek hotels with at least three stars.

Based on the requirements rules and the hotel offerings, we see that: (a) Hotel h1 is unacceptable because it does not provide business standard (rule r_2). (b) Hotel h4 is unacceptable because it does not have air-conditioning (rule r_4). (c) Hotel h6 is unacceptable because it does not have a gym (rule r_3). (d) Hotel h2 is unacceptable because its price (100) is higher than what Bob is willing to pay (90; rules r_8, r_9).e) Hotels h3, h5 and h7 are acceptable (rule r_1).

Table 2. Characteristics of Offered Hotels

Hotel	h1	h2	h3	h4	h5	h6	h7
Central	No	Yes	Yes	Yes	No	No	Yes
Public Transport	Yes	No	Yes	No	Yes	Yes	Yes
Gym	Yes	Yes	Yes	Yes	Yes	No	Yes
Pool	No	Yes	Yes	Yes	No	No	No
A/C	Yes	Yes	Yes	No	Yes	Yes	Yes
Category	2	4	3	3	3	3	3
Price	50	100	80	70	60	50	60

3.4.4 A Seller's Strategy

This case study concerns a seller's strategy that has been proposed by us.

The predicates we use are the following:

- step(S) is the step of the negotiation.
- now(T) is the current time.
- min_profit(X,W) is the minimum profit which seller seeks.
- cost(X,Y) is the cost Y of the product X.
- firstprice(X,Z) is the first offer of the seller for the product X.
- offer(X,S,Z) is the offer Z of seller for the product X at step S.
- counteroffer(X,S,Z) is the counteroffer Z of seller's opponent for product X at step S.

The strategy of the seller is the following:

r_0: $\Rightarrow$ negotiate

r_1: step(1), min_profit(X,W), cost(X,Y), firstprice(X,Z) $\Rightarrow$ offer(X,1, Y+W)

r_2: min_profit(X,W), immediate, W'=0.9W' $\Rightarrow$ min_proft(X,W')

r_3: step(S), now(T), event(t1,S-1,Z), Z=counteroffer, T-t1>30 $\Rightarrow \neg$negotiate

r_4: step(S), counteroffer(X,S-1,poso1), counteroffer(X,S,poso2), offer(s-1,Y1), poso2>poso1+20, Y=Y1-15 $\Rightarrow$ offer(X,S,Y)

r_5: step(S), counteroffer(X,S-1,poso1), counteroffer(X,S,poso2), poso2 $\leq$ poso1 $\Rightarrow \neg$negotiate

r_6: step(S),cost(X,Y),min_profit(X,W),counteroffer(X,S<Z),Z≤W+Y ⇒¬negotiate

r_7: step(S),min_profit(X,W),cost(X,Y),counteroffer(X,S,Y1),offer(X,S-1,Z),Y≥W+Y ⇒offer(X,S,0.98Z)

r_8: step(S),min_profit(X,W), cost(X,Y),counteroffer(X,S,Y1),Y1≥W+Y ⇒deal

r_9: now(T),T>tmax ⇒¬negotiate

$r_3, r_5, r_9 > r_0$
$r_1 > r_7$
$r_4 > r_7$

Rule r_1 states that at the start of negotiation the seller offers the value of the cost plus the value of the profit he desires. According to r_2 if the buyer declares from the beginning that he will pay immediately the seller reduces his first offer. In r_3, if 30 time units pass without the opponent making a counteroffer the negotiation terminates. Rule r_4 states that if buyer's current counteroffer is greater 20 units of money than his previous, the seller will lower his current offer by 15 units of money. According to r_5, if buyer's current offer is the same or smaller than his previous counteroffer the negotiation terminates. Rule r_6 states that if buyer's offer is less than products' cost plus the desired profit, the negotiation stops. Rule r_7 estimates the offer of the seller, which is 2% less than his previous offer. Rule r_8 states that if counteroffer is above the cost of the product plus the profit it is accepted. Finally r_9 terminates the negotiation if the time threshold expires.

3.4.5 An Agent which Participates in a Multi-attribute Negotiation

The last example regards a multi-attribute negotiation scenario and has been proposed by us. The domain of interest is that of PC selling. The negotiation does not concern only the price of the product but also other attributes, such as processor speed, memory size, monitor size, years of guarantee and possible gifts that accompany a PC. The predicates we use are the following:

- step(S) is the step of the negotiation.
- incr(S,I) is the amount by which buyer increases his offer at step S.
- cfp(S,X,P,M,Mon,Z,Y) is a predicate which represents a complete offer for a product X at step S, with the attributes, P, M, Mon, Z, Y except for the price.
- negotiate(X) defines the state of the negotiation.

- buyable(X) defines if a product is buyable according to user's preferences.
- buyable(X,processor), buyable(X,memory) and buyable(X,monitor) indicate if a products is buyable or not due to one of its attributes.
- propose(X,Bid) defines a proposal Bid, for a product X .
- accept (X,Bid) indicates the acceptance of a proposal for product X.
- deal(X,Bid) indicates that a deal has been achieved.

We model the strategy of both buyer and seller using an arbitrary negotiation protocol for multi-attribute negotiation. According to the protocol: (a) the buyer and seller initially agree on the attributes and consequently on the price of the product. (b) The seller initiates the negotiation by proposing a set of attributes for a product. (c) The step of the negotiation increases for both the participants when they have sent a message and received a response. (d) We make the convention that buyer only sends propose and buyable or ¬buyable messages, while seller only sends cfp and accept or ¬accept messages.

We firstly examine the rules of the buyer. Rule r_0 is the general case and the product is considered to be buyable. Rule r_1 defines that the increment is 0 when negotiation starts while r_2 defines that increment increases by 40 at each step of the protocol.

r_0: $\Rightarrow$ buyable(X)

r_1: step(S), S=1, I=0 $\Rightarrow$ incr(I)

r_2: step(S),S>1,incr(S-1,I) $\Rightarrow$ incr(S,I+40)

r_3: cfp(S,X,P,M,Mon,Z,Y) $\Rightarrow$ negotiate(X)

r_4: cfp(S,X,P,M,Mon,Z,Y),P<1500 $\Rightarrow$ ¬buyable(X,processor)

r_5: cfp(S,X,P,M,Mon,Z,Y),M<256 $\Rightarrow$ ¬buyable(X,memory)

r_6: cfp(S,X,P,M,Mon,Z,Y),Mon$\leq$15 $\Rightarrow$ ¬buyable(X,monitor)

r_7: cfp(S,X,P,M,Mon,Z,Y),1200$\leq$P$\leq$1500, M$\geq$800 $\Rightarrow$ buyable(X)

r_8: buyable(X),incr(S,I) $\Rightarrow$ propose(X,700+I)

r_9: buyable(X),incr(S,I),gift(X,Z),Z="printer"$\Rightarrow$ propose(X,800+I)

r_{10}: buyable(X),incr(S,I),guarantee(X,Y), Y$\geq$2 $\Rightarrow$ propose(X,900+I)

r_{11}: propose(X,Bid),Bid>900, $\Rightarrow$ ¬negotiate(X)

r_{12}: accept(X,Bid) $\Rightarrow$ deal(X,Bid)

r_{13}: step(S),S=7 $\Rightarrow$ ¬negotiate(X)

r_{14}: ¬negotiate(X) $\Rightarrow$ ¬propose(X)

$r_4, r_5, r_6 > r_0$
$r_1 > r_2$

r_{11}, r_{13} > r_3
r_7 > r_4, r_5
r_6 > r_7
r_9, r_{10} > r_8
r_{10} > r_9

According to rule r_3, if there is any proposal by the opponent, the negotiation begins. Rules r_4 to r_7 model buyer's preferences for the separate attributes of the product. In particular, if the pc has speed less than 1500 MHz, memory less than 256 MB or monitor smaller or equal to 15 inches, it is not appropriate for the buyer. In these three cases, a corresponding predicate is sent to the opponent to inform him which attribute needs change. R_8 fires when the product is buyable. Rule r_7 contradicts rule r_4 and defines that if the pc has a memory more than 800 MB it is acceptable no matter its speed is lower. Accordingly, rule r_7 is superior to rules r_4, r_5. Rules r_8 r_9 and r_{10} propose a price for the product in different cases. Rule r_8 is the general case and the offer increases by the increment. Rules r_9 and r_{10} propose a greater offer when there is a gift or the guarantee is valid for more than two years. R_{11} states that if buyer proposes an amount greater than 900 money units, the negotiation must terminate. According to rule r_{12}, if buyer receives an accept he sends a deal to his opponent. Rule r_{13} defines that the negotiation should have been completed in 7 steps. Finally, rule r_{14} states that if the negotiation has been terminated no proposal should be submitted.

conflict::cfp(S+1,X,P+100,M,Mon,Z,Y), cfp(S+1,X,P,M+100,Mon,Z,Y), cfp(S+1,X,P,M,Mon+2,Z,Y), cfp(S+1,X,P,M,Mon,Z,Y+1)

r_1: ⇒ negotiate(X)

r_2: ⇒ cfp(S,X,P,M,Mon,Z,Y)

r_3: ¬buyable(X,Processor),cfp(S+1,X,P,M,Mon,Z,Y)
⇒ cfp(S+1,X,P+100,M,Mon,Z,Y)

r_4: ¬buyable(X,Memory),cfp(S,X,P,M,Mon,Z,Y)
⇒ cfp(S+1,X,P,M+100,Mon,Z,Y)

r_5: ¬buyable(X,Monitor),cfp(S,X,P,M,Mon,Z,Y)
⇒ cfp(S+1,X,P,M,Mon+2,Z,Y)

r_6: ¬buyable(X,Var),cfp(S,X,P,M,Mon,Z,Y),step(S),S≥4
⇒ cfp(S+1,X,P,M,Mon,Z,Y+1)

r_7: propose(X,Bid),cfp(S,X,P,M,Mon,Z,Y),Price=
0.3*P+0.2*M+10*I+20*Y, Bid ≥ Price ⇒ accept(X,Bid)

r_8: propose(X,Bid),cfp(S,X,P,M,Mon,Z,Y),Price=
0.3*P+0.2*M+10*I+20*Y,Bid < Price ⇒ ¬accept(X,Bid)

r_9: step(S),S=6 ⇒ ¬negotiate(X)

```
r9 > r1
r3, r4, r5, r6 > r2
r6 > r3, r4, r5
r5 > r3, r4
r4 > r3
```

At this point we analyze the rules of the seller. Initially seller issues a proposal according to the stored facts. According to rules r_3, r_4, r_5, if buyer reject the product, the values of the attributes, which correspond to processor speed, memory size and monitor size change and a new call for proposal is submitted. Superiority relation `r5 > r3, r4` defines that if a seller receive two or more predicates corresponding to different reasons of whether a product is not buyable, he will firstly seek to increase the monitor size. Rule r_6 is quite similar but the years of guarantee change this time. Rules r_7 and r_8 are the most important for the strategy of the seller. If buyer has issued a proposal for product X for a certain combination of attributes, a function evaluates the proposal and either an accept or not accept message is issued and returned to the buyer.

Now let's see how the two defeasible theories interact. We suppose that initially the step of negotiation is 0 and seller issues the following call for proposal: cfp(0,pcx1,1400,400,15,"",1). That means that the negotiation is about a PC with code pcx1 the step is 0, the PC has a processor of 1400 MHz, a memory of 400 MB, a monitor of 15' and there is 1 year of guarantee. We see that: (a) buyer sends back two messages `¬buyable(X,processor), ¬buyable(X,monitor)` (rules r_4, r_6). The step of negotiation is now 1. (b) Seller in turn submmits a cfp(0,pcx1, 1400,400,17,"",1) (rule r_5) and buyer now sends a `¬buyable(X,processor)`. The step of negotiation is now 2. (c) Seller submits cfp(0,pcx1,1500,400,17,"",1) (rule r_3) and buyer sends back a propose (X,780) (rule r_8). The increment is 80 because two negotiation steps have already passed. Finally as the negotiation step is 3 seller sends an accept(X,780) (rule r_7) and buyer responds with a deal(X,780) (rule r_{12}).

4. Implementation

The purpose of this section is an analytic presentation of the implementation and an in-depth examination of our proposed solution. We start by presenting JADE along with its most important characteristics.

Subsequently, we deal with the implementation of the negotiating agent. We firstly present the architecture of the negotiating agent, which is based on defeasible logic for the expression of the negotiation strategy. We then present the modeling of a simple negotiation domain by means of message content ontology. Afterwards we describe in detail the implemented 1-1 negotiation protocol and justify its use. We fabricate a negotiation scenario and in turn model a strategy for the chosen 1-1 negotiation protocol, by means of defeasible logic rules. Lastly, we provide a detailed trace of the interactions, between the negotiating agents.

The next part of this section concerns the implementation of a brokering system. We initially present the overall architecture of the system and describe the modules of the architecture along with the interactions and actions that are performed during a brokering case. We then present the modeling of a simple brokering domain by means of message content ontology. As in the case of brokering the service or the product are of great importance we model it along with its characteristics in separate domain ontology, using RDF-RDFS technology. Subsequently, we describe the protocol that is used among the agents, which participate in brokering. As a next step, we fabricate a brokering scenario and in turn present the formalization of user's requirements-preferences and the representation of the offers. Lastly, we provide a detailed trace of the interactions, between the agents that participate in the brokering scenario.

The last part of this section regards the limitations of our implementation along with the choices and convention we had to make.

4.1 Multiagent Framework - JADE

For the development of our negotiating agent architecture and the brokering system we chose JADE [35], [68]. According to [70], it exhibits very interesting features compared to other multiagent system frameworks. In addition, a review on the

literature of multiagent applications reveals an increasing popularity and acceptance of JADE.

From the functional point of view, JADE provides the basic services necessary to distributed peer-to-peer applications in the fixed and mobile environment. JADE allows each agent to dynamically discover other agents and to communicate with them according to the peer-to-peer paradigm. From the application point of view, each agent is identified by a unique name and provides a set of services. It can register and modify its services and/or search for agents providing given services, it can control its life cycle and, in particular, communicate with all other peers.

Each running instance of the JADE runtime environment is called a *Container* as it can contain several agents. The set of active containers is called a *Platform*. A single special *Main Container* must always be active in a platform and all other containers register with it as soon as they start. It follows that the first container to start in a platform must be a main container while all other containers must be “normal” (i.e. non-main) containers and must “be told” where to find (host and port) their main container (i.e. the main container to register with). JADE uses RMI technology for intra-platform communication and CORBA or HTTP technology for inter-platform communication.

Besides the ability of accepting registrations from other containers, a main container differs from normal containers as it holds two special agents who are created automatically. The AMS (Agent Management System) that provides the naming service (i.e. ensures that each agent in the platform has a unique name) and represents the authority in the platform (for instance it is possible to create/kill agents on remote containers by requesting that to the AMS) and the DF (Directory Facilitator) that provides a Yellow Pages service by means of which an agent can find other agents providing the services he requires in order to achieve his goals.

Fig.15 illustrates the above concepts through a sample scenario showing two JADE platforms composed of 3 and 1 container respectively. JADE agents are identified by a unique name and, provided they know each other’s name, they can communicate transparently regardless of their actual location: same container (e.g. agents A2 and A3 in Figure 15), different containers in the same platform (e.g. A1 and A2) or different platforms (e.g. A4 and A5).

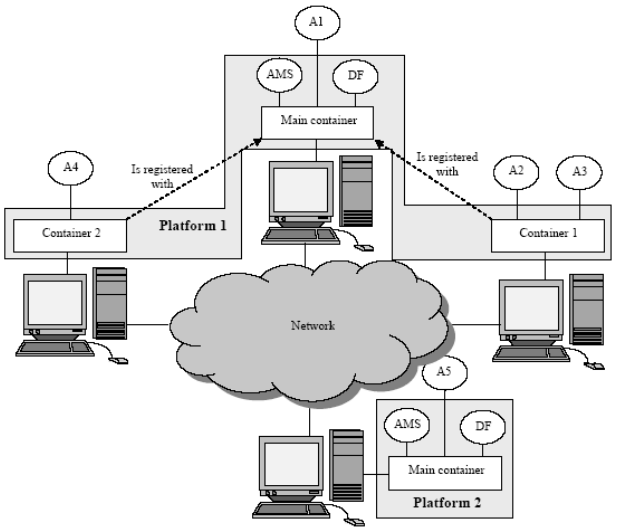


Figure 15. JADE Platform

JADE Agents communicate by exchanging asynchronous messages, a communication model almost universally accepted for distributed and loosely coupled communications, i.e. between heterogeneous entities that do not know anything about each other. In order to communicate, an agent just sends a message to a destination. Agents are identified by a name (no need for the destination object reference to send a message) and, as a consequence, there is no temporal dependency between communicating agents. The sender and the receiver could not be available at the same time. The receiver may not even exist (or not yet exist) or could not be directly known by the sender that can specify a property (e.g. "all agents interested in football") as a destination. Because agents identify each other by their name, hot change of their object reference are transparent to applications.

Despite this type of communication, security is preserved, since, for applications that require it, JADE provides proper mechanisms to authenticate and verify "rights" assigned to agents. When needed, therefore, an application can verify the identity of the sender of a message and prevent actions not allowed to perform (for instance an agent may be allowed to receive messages from the agent representing the boss, but not to send messages to it). All messages exchanged between agents are carried out within an envelope including only the information required by the transport layer. That allows, among others, to encrypt the content of a message separately from the envelope.

The structure of a message complies with the ACL language defined by FIPA [73] and includes fields, such as variables indicating the context a message refers-to and

timeout that can be waited before an answer is received, aimed at supporting complex interactions and multiple parallel conversations. To further support the implementation of complex conversations, JADE provides a set of skeletons of typical interaction patterns to perform specific tasks, such as negotiations, auctions and task delegation. By using these skeletons (implemented as Java abstract classes), programmers can get rid of the burden of dealing with synchronization issues, timeouts, error conditions and, in general, all those aspects that are not strictly related to the application logic. To facilitate the creation and handling of messages content, JADE provides support for automatically converting back and forth between the format suitable for content exchange, including XML and RDF, and the format suitable for content manipulation (i.e. Java objects). This support is integrated with some ontology creation tools, e.g. Protégé, allowing programmers to graphically create their ontology.

JADE is opaque to the underlying inference engine system, if inferences are needed for a specific application, and it allows programmers to reuse their preferred system. It has been already integrated and tested with JESS and Prolog.

To increase scalability or also to meet the constraints of environments with limited resources, JADE provides the opportunity of executing multiple parallel tasks within the same Java thread. Several elementary tasks, such as communication, may then be combined to form more complex tasks structured as concurrent Finite States Machines. In the J2SE and Personal Java environments, JADE supports mobility of code and of execution state. That is, an agent can stop running on a host, migrate on a different remote host (without the need to have the agent code already installed on that host), and restart its execution from the point it was interrupted (actually, JADE implements a form of not-so-weak mobility because the stack and the program counter cannot be saved in Java).

The platform also includes a naming service (ensuring each agent has a unique name) and a yellow pages service that can be distributed across multiple hosts. Federation graphs can be created in order to define structured domains of agent services.

Another very important feature consists in the availability of a rich suite of graphical tools supporting both the debugging and management/monitoring phases of application life cycle. By means of these tools, it is possible to remotely control agents, even if already deployed and running: agent conversations can be emulated, exchanged

messages can be sniffed, tasks can be monitored, and agent life-cycle can be controlled.

The described pieces of functionality, and particularly the possibility of remotely activating (both from code and from console), even on mobile terminals, tasks, conversations and new peers, makes JADE very well suited to support the development and execution of distributed, machine-to-machine, multi-party, intelligent and proactive applications.

4.2 Negotiating Agent Implementation

In this section we present an architecture for a negotiating agent, along with a small negotiation locale where the negotiation takes place. We identify two roles, the buyer and the seller. There is no admission policy. The negotiating agents interact directly without any intervention by 3rd parties. We assume that agents know the addresses of each other. Our demonstration is illustrated by 1-1 negotiation or single bargaining and concerns only one product and one attribute of that product which is the price. The protocol is hard-coded to the agents. Involved parties share a negotiation template that specifies the different parameters of the negotiation like price, product type, product colour etc. All the parameters are fixed except for the price, which is subject to negotiation. For the definition of the template, an ontology in protégé is used. The strategy of the agents is expressed by defeasible logic rules, and can be easily loaded to the agents, leading to a plug-and-play architecture.

4.2.1 Negotiating Agent Architecture

The Negotiating *Agent Architecture* we implemented was primarily based on the architecture proposed by Dumas et al. [10]. Software agents consists of four components: (a) A memory which contains past decisions and interactions (Knowledge Base), (b) A communication module which handles incoming and outcoming messages (JADE platform), (c) A reasoning module (DR-DEVICE Inference Engine) (d) A control module for the coordination of the above components (script in Java).

The architecture of the negotiating agent is depicted in Fig.16. When the agent is notified of an external event, such as an incoming message (step 1), the control module initially retrieves a fact template from the local storage unit (step 2) and consequently, the negotiation parameters from the memory (step 3). The template is an empty placeholder in line with DR-DEVICE system syntax. When the template is filled with the negotiation parameters, is then regarded as “the facts”. The control module updates

the knowledge base with the new facts (step 4) and then activates DR-DEVICE (step 5). DR-DEVICE in turn retrieves from the knowledge base the facts, along with the strategy (step 6) and starts the inference process. After the inference has been completed, the knowledge base is updated with the results (step 7). The control module queries the knowledge base for the result (step 8) and after a short processing; an appropriate message is posted to the communication module.

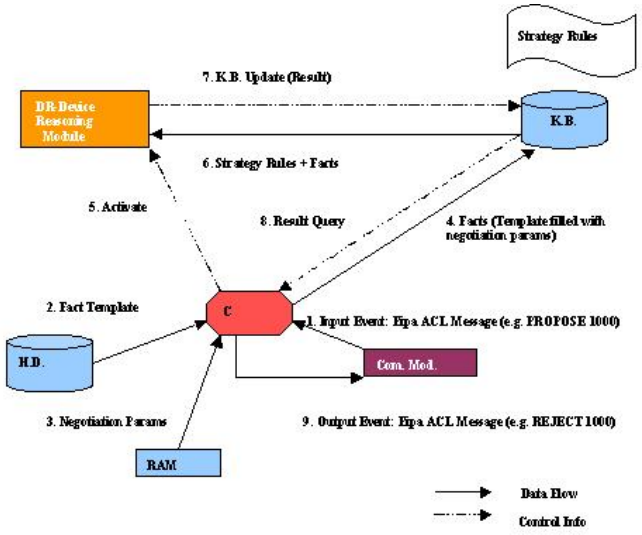


Figure 16. Negotiating Agent Architecture

4.2.2 The Negotiation Protocol

As we have already stressed, a basic condition for the automation of the negotiation process among intelligent agents is the existence of a negotiation protocol that encodes the allowed sequences of actions, or in other words the rules of the game. Our first thought was to use a well-defined protocol for 1-1 automated negotiation. Although FIPA provides a plethora of standardized protocols, such as FIPA brokering, FIPA English auction, FIPA Contract net etc., we found that there is no standard interaction protocol when it comes to 1-1 automated negotiation. As a result, we implemented a negotiation protocol proposed in [22].

This protocol is a finite state machine that must be hard-coded to all agents, participating into the negotiation. Bartolini et al. [6] say that most multi-agents systems today use a single negotiation protocol, which is usually a finite state machine, hard-coded to all the agents, leading to an inflexible environment, which can accept only agents designed for it. To overcome this inflexibility they propose a generic interaction protocol, which can be parameterized with different negotiation rules and give different

negotiation mechanisms. The rules can be exchanged among agents that are able to inform their peers, which protocol they wish to use. However, the focus of our work is not to protocol design and we believe the protocol we use is a good solution for our demonstrator.

As we have already said, our protocol is a finite state machine with discrete states and transition. The protocol is depicted in Fig.17. S0 to S6 represent the states of a negotiation and E is the final state in which there is an agreement, or a failure of agreement between the participants. Send and Recv predicates represent the interactions that cause state transitions. To clarify the function of the protocol we give an example. If the sequence of transitions is the following: S0→ S1→S2→S6→E, that means that the agent initially sends a call for proposal message (CFP) to the other negotiating agent (S0→ S1), then he receives a propose message (S1→S2) and after the evaluation he decides to send an accept message (S2→S6). Lastly he receives an accept message and the negotiation terminates successfully (S6→E). We make the convention that the participant that plays the role of the buyer starts the negotiation by posting a CFP message. So, while the protocol can be used as it is by a buyer, it needs a small modification for a seller. Particularly instead of the transition S0→ S1 there should be a transition S0→ S2 with label "Recv CFP".

4.2.3 The Negotiation Strategy

The strategy of a potential buyer or seller during a negotiation scenario is very critical for the outcome of the encounter. Every strategy is indeed designed in line with a particular protocol. As we have already seen, there is a plethora of strategies classified according to different criteria. We based the strategies we used on the work of Tsang et al. [23]. They define the simple constrained bargaining game between one buyer and one seller. Some of the most important assumptions are that the seller is constrained by the cost and the number of days within which he has to sell, while the buyer is constrained by his utility and the number of days within which he has to buy. In addition, neither the buyer nor the seller has information about the constraints of the other. The players make sequential bids with the seller to bid first and they can bid only once per day. An agreement is reached when both buyer and seller bid for the same price. Finally, according to the assumptions, if a deal cannot be made before a player runs out of time the negotiation terminates.

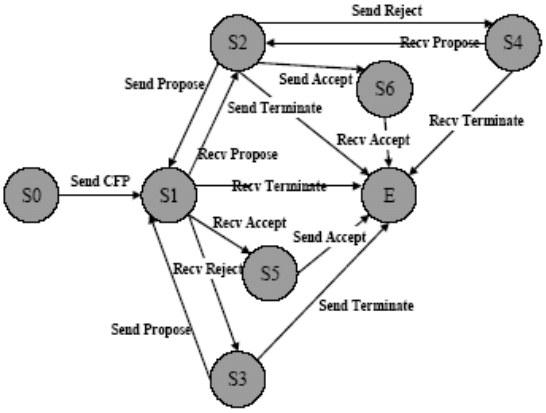


Figure 17. 1-1Negotiation Protocol

Tsang et al. propose a number of different strategies both for buyer and sellers. For our buyer we adopted the simple buyer strategy, whose characteristics are described in Table 3. We have made the following changes for the strategy of the buyer: *Firstly*, buyer and seller are not obliged to make only one offer per day but one offer per negotiation step. Negotiation step is handled by the protocol and increases each time a player (buyer or seller) has made an offer and subsequently has received a counteroffer or another message. So, we speak about time to buy (TTB) and time to sell (TTS), measured in negotiation steps. *Secondly*, except for the offer-acceptance criterion we have added a check during the offer submission to avoid results, which are against the benefit of the player. *Thirdly*, we incorporated into the strategy parameters relevant to the protocol like the state of the negotiation and the step of the negotiation. *Lastly*, an agreement is reached when both buyer and seller send an "agree" message.

Table 3. Characteristics of Buyer's Negotiation Strategy

Strategy Name	First Bid Algorithm	Offer-Acceptance Criterion	Last Day Bidding	General Bid Algorithm
Simple Buyer	Utility/DTB	Counteroffer +Minimum Profit< Utility	As usual	Bid half way between previous bid price and utility

For the buyer, participating into the negotiation, we used the modified strategy of Tsang et al. and we expressed it in defeasible logic. For the seller we used a strategy hard-coded in java to demonstrate that agents with different architecture can interact without any problems. Seller's strategy is quite similar with that of buyer, except for

the general bidding strategy. Seller decreases his offer by a fixed amount while buyer increases his offer in a linear fashion.

We express the buyer's strategy in defeasible logic as depicted in Fig 18. The predicates we use are the following:

- Step(s): The step of the negotiation. When a buyer or seller sends a message and then receives another one the step is increased by one.
- Counteroffer(c): The offer which a buyer or seller receives from the opponent.
- Min_profit(mp): The minimum profit the buyer seeks after buying the product.
- Utility(u): The utility of the buyer if he buys the product.
- Ttb(ttb): The time (negotiation steps) the buyer has at his disposal in order to buy the product.
- State(st): The current state of the negotiation according to the protocol. The possible states are:

  1(S1)→The buyer has already sent a CFP or a PROPOSE message.

  2(S2)→ The buyer has already received a PROPOSE message.

  …
- First_bid(fb): The initial bid of the buyer.
- Previous_bid(prb): The previous bid of the buyer.

Rules r_1, r_2, and r_3 define the conditions for the acceptance or rejection of a proposal. More specific rule r_1 states that if the current state of the negotiation is S2 (agent has received a "propose" message) and if opponent's offer plus the minimum profit is less or equal to half the utility, the counteroffer is accepted in all cases.

r_1: State(st),Counteroffer(c),Min_profit(mp),Utility(u),st=2,c+mp≤u/2
⇒ ACCEPT_PROPOSAL
r_2: State(st),Counteroffer(c),Min_profit(mp),Utility(u), st=2,c+mp>u
⇒ ~ACCEPT_PROPOSAL
r_3: State(st),st=5 ⇒ ACCEPT_PROPOSAL
r_4: Step(s),Counteroffer(c),Min_profit(mp),Utility(u),First_Bid(fb),
State(st),s=0,st=2,u/2<c+mp≤u,bid=fb ⇒ PROPOSE(bid)
r_5: Step(s),Ttb(ttb),State(st),Counteroffer(c),Min_profit(mp),Utility(u),
First_Bid(fb),Previous_bid(prb),0<s≤ttb-1,st=2,u/2<c+mp≤u,prb=0,bid=fb
⇒ PROPOSE(bid)
r_6: Step(s),Ttb(ttb),State(st),Counteroffer(c),Min_profit(mp),Utility(u),Previ
ous_bid(prb),0<s<ttb-1,st=2,u/2<c+mp≤u,prb!=0,bid=(u-prb)/2+prb
⇒ PRELIM_PROPOSE(bid)
r_7: Step(s),Ttb(ttb),State(st),Previous_bid(prb),Utility(u),0<s<ttb,
st=3,bid=(u-prb)/2+prb ⇒ PRELIM_PROPOSE(bid)
r_8: PRELIM_PROPOSE(bid) ⇒ PROPOSE(bid)
r_9: Min_profit(mp),Utility(u), PRELIM_PROPOSE(bid),bid>u-mp ⇒ ~PROPOSE(bid)
r_{10}: Min_profit(mp),Utility(u),PRELIM_PROPOSE(bid),bid>u-mp,new_bid=utility-
min_profit ⇒ PROPOSE(new_bid)

$r_9 > r_8$

Figure 18. Buyer's Strategy in Defeasible Logic

r_2 describes the case in which opponent's offer plus the minimum profit is greater than his utility and the counteroffer is rejected. Finally r_3 defines that if the current state of the negotiation is S5 (agent has received an "accept" message) he also sends an "accept" message. There are two levels for the bidding policy. Bidding of first step and general bidding policy.

r_4 states that if the negotiation is at state S2 and at the first step, the utility divided by the ttb is offered. According to r_5 if the current state of the negotiation is S2 (Agent has received a "propose" message) *but* it has not made one, it offers the utility divided by the ttb.

r_6 defines that if the current state of the negotiation is S2 (agent has received a "propose" message) *and* it has made an offer in the previous step, it increases linearly its offer, which derives from the type:

$$bid = \frac{utility - previous\_bid}{2} + previous\_bid$$

r_7 describes that if the current state of the negotiation is S3 (agent has received a "reject" message) it also offers the above bid.

r_8 defines that if r_7 or r_6 is true then the computed amount for the bid is to be offered. However, r_9 checks if the bid to be offered is lower than the utility minus the minimum profit and if it is not, r_{10} is fired. Rule r_9 is an additional check that ensures that the offered amount of money for the product is not against the benefit of the buyer.

4.2.4 Message Content Ontology for Negotiation

As we have already said, Message Content Ontology (M.C.O.) is a special purpose ontology, which allows agents to model facts, beliefs, allowed actions, hypotheses and predications about a domain. We used an ontology particularly for actions, that agents can request and perform during the negotiation procedure and predicates that are error messages. There are also defined a few characteristics of a product. The unique action of our M.C.O is: Suggest. We used Protégé [69] for the design of the ontology and Beangenerator, a tool that transforms Ontologies defined in protégé model or RDFS into java classes, for easiest manipulation by the agents.

4.2.5 Negotiation Trace

In this section we demonstrate the operation of the system and we examine a negotiation trace between a buyer agent and a seller agent. JADE platform provides a special-purpose agent that is called sniffer. Sniffer can monitor the exchanged messages of two or more agents in the agent platform. The specific parameters of the negotiation are given in the next table. We examine the trace from the point of view of the buyer. The parameters of the negotiation are summarized in Table 4.

Table 4. Negotiation Trace Parameters

BUYER	SELLER
Ttb=5	Tts=10
Minimum Profit = 100	Minimum Profit = 100
Utility = 1000	Maximum Profit = 800
	Bid decrement = 40
	Cost = 200

As we can see in Fig. 19, the buyer initially issues a "Call for Proposal" message (CFP). At this point, as it is the first time we present a trace, we analyse the structure of exchanged messages. As we have already said in section 2.1.4, FIPA ACL messages are built up of three layers of languages. (a) Elements of the world are defined in an ontology. (b) An agent's intention to describe or alter the world is expressed by a communicative act or speech-act such as INFORM and (c) statements about the world are expressed by means of a Content Language. In order for agents to be able to reason about the effects of their communication, ACL messages are inserted into proper Agent Interaction Protocols that describe allowed sequences of actions among agents.

At the left-hand side of Fig.19 one can see all the interactions between the buyer and the seller agent. We analyze the first interaction. The ACL message that corresponds to interaction 1 is depicted next to the interactions and is indicated by the red array. The *Communicative Act* (or Speech-Act) of this ACL message is "CFP". The *Ontology,* which both buyer and seller share, is called "Negotiation" and the used *Interaction* protocol (or Negotiation Protocol) is called "Simple-Bargaining". When it comes to the content language, we use FIPA SL0. The choice of the content language is justified in section 4.4.2. However, for the first interaction we also show the content of the message in FIPA-RDF content language (Fig. 20) to stress that different content languages can be used for the same negotiation scenario.

The content of the message is indicated by the blue arrow. According to the content of the message, the *Actor* of the action is Buyer who suggests a negotiation template, about a single item of a black NOKIA 1100 mobile phone.

Lastly we must say that JADE encodes ACL messages in XML before transmission. So the content of the message is finally a SL0 or FIPA-RDF expression, wrapped in an XML message. The encoding of the message to XML is transparent to JADE developer.

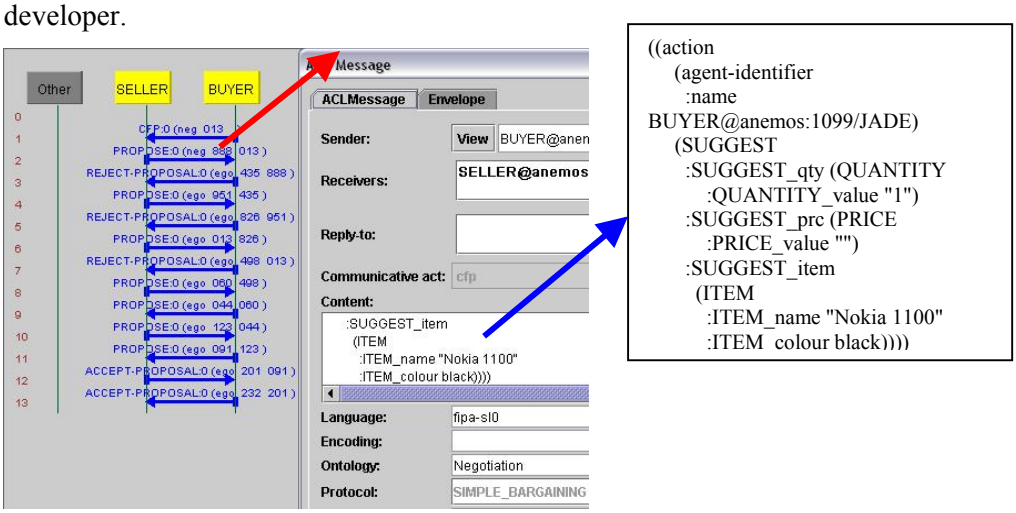


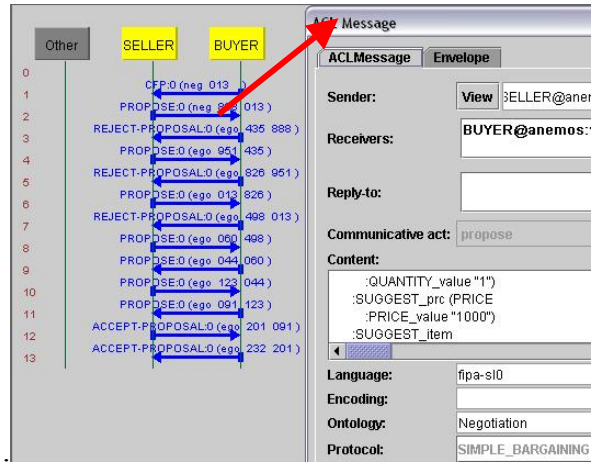
Figure 19. Negotiation ACL message 1 - Buyer issues a "Call for Proposal"

The seller responds with a "Propose". The amount proposed by the seller is 1000, as it can be seen from the content of the message (Fig. 21). According to the rule R2 of buyer's strategy, as long as the relation c+mp > u is true, the buyer keeps rejecting the offer (Fig. 22).

```
<?xml version="1.0"?>
 <rdf:RDF
            xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#"
            xmlns:fipa-rdf="http://www.fipa.org/schemas/FIPA-RDF#"
        xmlns:Negotiation="http://www.csd.uoc.gr/~dogjohn#">
 <rdf:object>
  <fipa-rdf:CONTENT_ELEMENT>
    <rdf:Description>
      <fipa-rdf:type>action</fipa-rdf:type>
      <Negotiation:actor>
        <rdf:Description>
            <Negotiation:name>BUYER@anemos:1099/JADE</Negotiation:name>
        </rdf:Description>
      </Negotiation:actor>
      <Negotiation:action>
        <rdf:Description>
          <fipa-rdf:type>SUGGEST</fipardf:type>
          <Negotiation:SUGGEST_prc>
            <rdf:Description>
              <Negotiation:PRICE_value></Negotiation:PRICE_value>
            </rdf:Description>
          </Negotiation:SUGGEST_prc>
          <Negotiation:SUGGEST_qty>
            <rdf:Description>
              <Negotiation:QUANTITY_value>1</Negotiation:QUANTITY_value>
            </rdf:Description>
          </Negotiation:SUGGEST_qty>
```

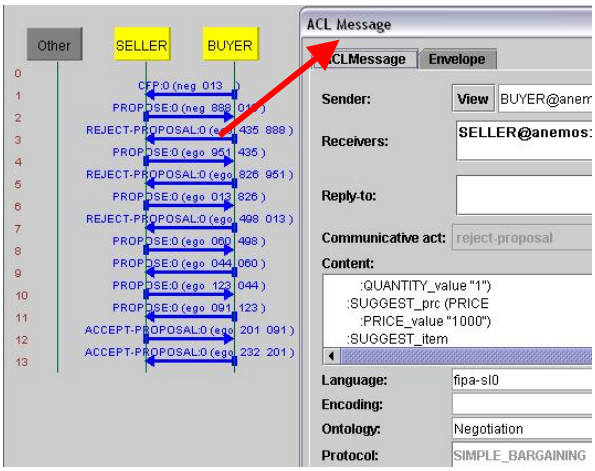
```
            <Negotiation:SUGGEST_item>
                <rdf:Description>
                  <Negotiation:ITEM_colour>black</Negotiation:ITEM_colour>
                  <Negotiation:ITEM_name>Nokia 1100</Negotiation:ITEM_name>
                </rdf:Description>
             </Negotiation:SUGGEST_item>
          </rdf:Description>
       </Negotiation:action>
    </rdf:Description>
 </fipa-rdf:CONTENT_ELEMENT>
</rdf:object>
</rdf:RDF>
```

Figure 20. ACL message in FIPA-RDF Format




```
((action
   (agent-identifier
     :name
SELLER@anemos:1099/JADE
     :addresses
      (sequence
IOR:000000000000001149444C3A46495
0412F4D54533A312E300000000000000
0010000000000000060000102000000000
A3132372E302E302E31000BE00000001
9AFABCB000000000214BEB630000000
0800000000000000000A0000000000000
1000000010000002000000000000100010
0000002050100010001002000010109000
0000100010100))
   (SUGGEST
     :SUGGEST_qty (QUANTITY
       :QUANTITY_value "1")
     :SUGGEST_prc (PRICE
       :PRICE  value "1000")
```

Figure 21. Negotiation ACL message 2 - Seller proposes 1000




```
((action
   (agent-identifier
     :name
BUYER@anemos:1099/JADE)
   (SUGGEST
     :SUGGEST_qty (QUANTITY
       :QUANTITY_value "1")
     :SUGGEST_prc (PRICE
       :PRICE_value "1000")
     :SUGGEST_item
      (ITEM
       :ITEM_name "Nokia 1100"
       :ITEM  colour black))))
```

Figure 22. Negotiation ACL message 3 - Buyer rejects 1000

As the buyer has rejected the seller's offer, at the next step of the negotiation, the seller decreases its offered amount by 40 (bid decrement) and waits for the response of the buyer (Fig. 23)

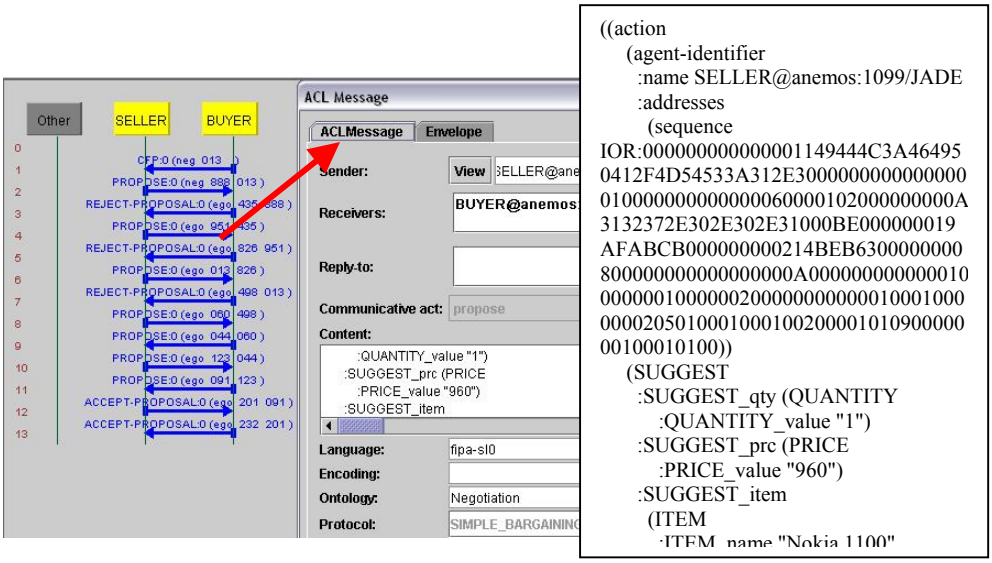


Figure 23. Negotiation ACL message 4 - Seller proposes 960

As the buyer regards (according to his strategy) that the amount of 960 is too high, he continues to reject the offer, without issuing a counteroffer (Fig.24). At this point we must say that although the protocol allows both for a counteroffer or a rejection of a proposal, the decision lies with the agent and is expressed through the strategy.

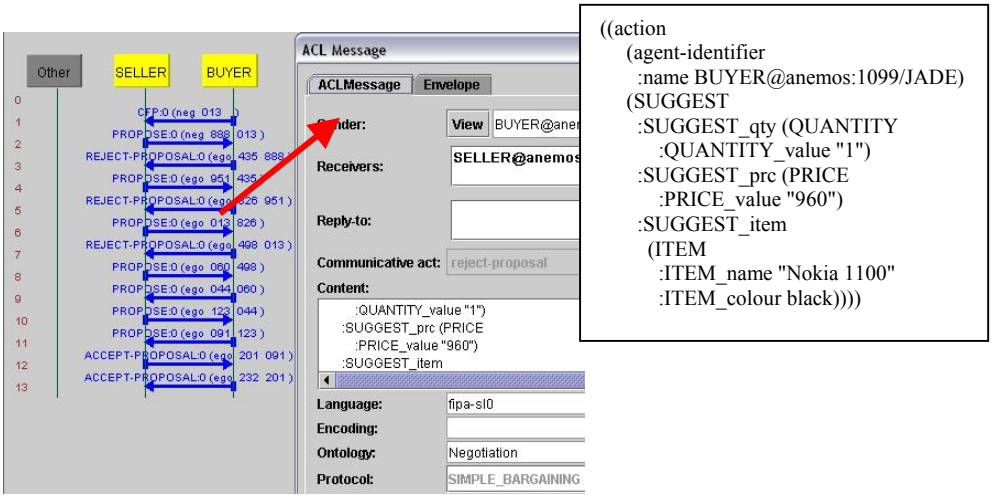


Figure 24. Negotiation ACL message 5 - Buyer rejects 960

The seller decreases his offer by another 40, offering 920 (Fig.25). The buyer still rejects seller's offer (Fig.26) and the latter subsequently offers 880 (Fig.27).

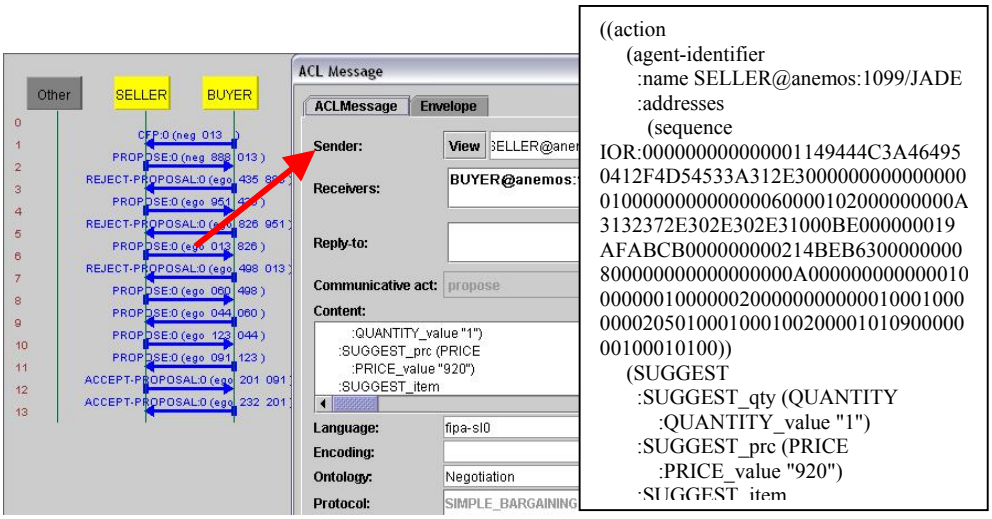


Figure 25. Negotiation ACL message 6 - Seller proposes 920

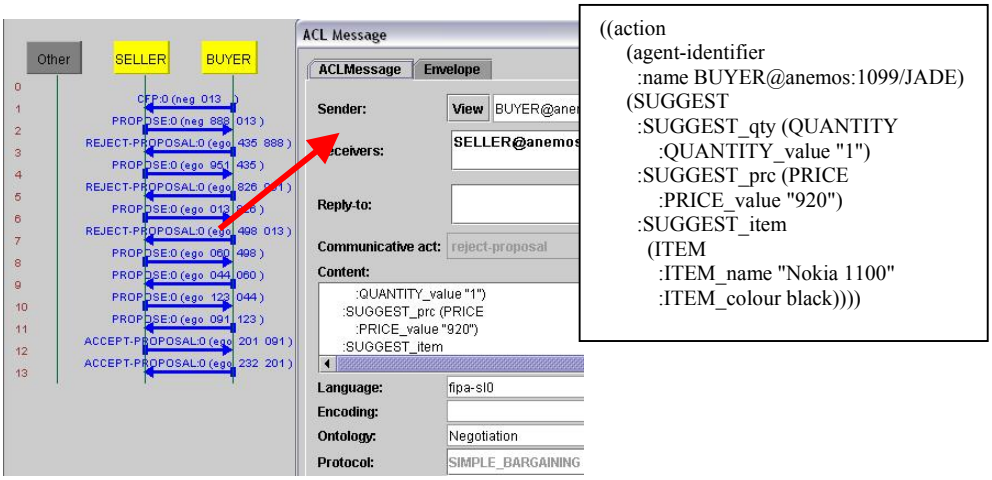


Figure 26. Negotiation ACL message 7 - Buyer rejects 960

As the relation 880+100>1000 (c+mp>u) is now false, R5 fires and the buyer offers his initial offer, which is the amount 1000/5 (u/ttb). This is depicted in Fig.28.

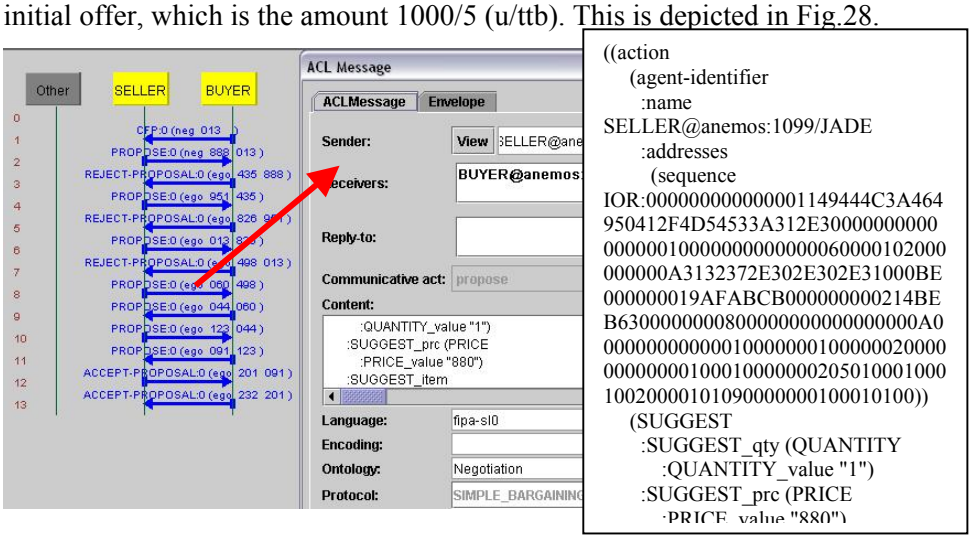


Figure 27. Negotiation ACL message 8 - Seller proposes 880

At the next step, the seller offers 840 and waits the buyer for its response (Fig. 29). Rule R6 now fires and buyer offers the amount 600 (Fig. 30).

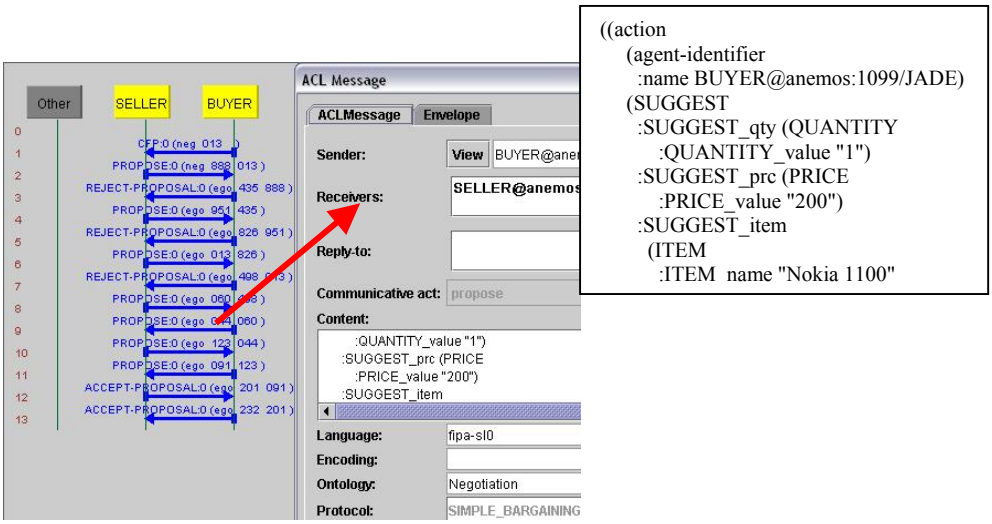


Figure 28. Negotiation ACL message 9 - Buyer proposes 200

The seller in turn issues an “Accept Proposal” (Fig. 31) message and the negotiation terminates. At this point we must notice that if seller’s last message were not “Accept Proposal”, the buyer’s control module would issue a “Cancel” as the ttb would exceed 5.

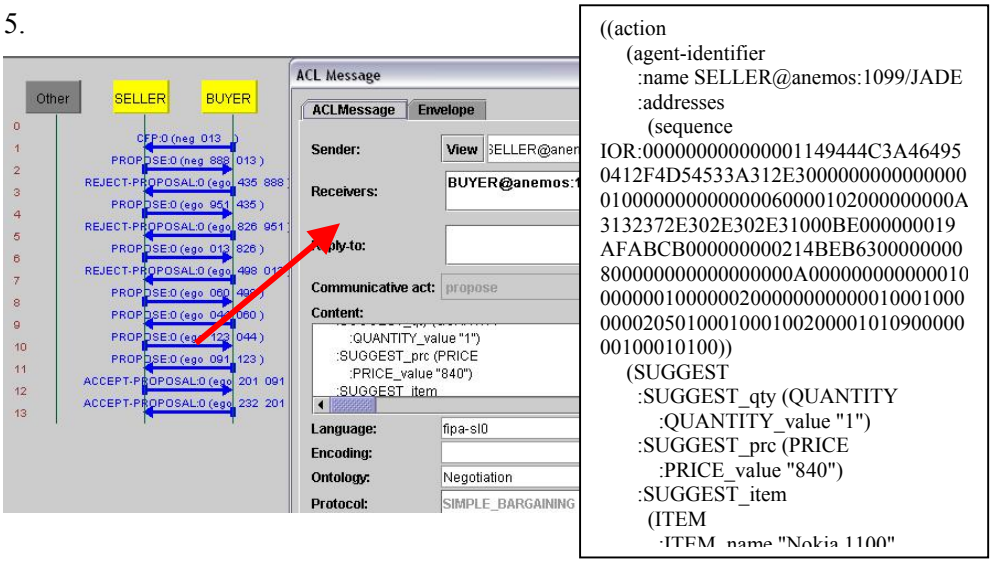


Figure 29. Negotiation ACL message 10 - Seller proposes 840

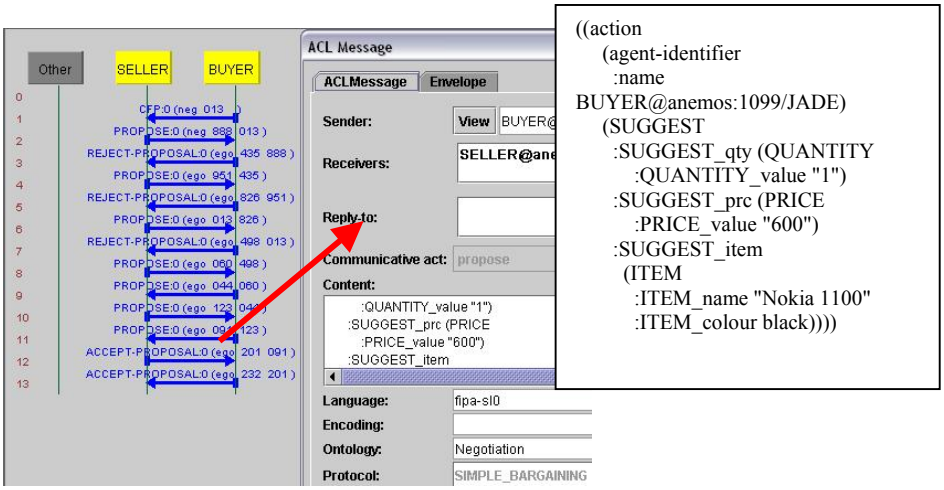


Figure 30. Negotiation ACL message 11 - Buyer proposes 600

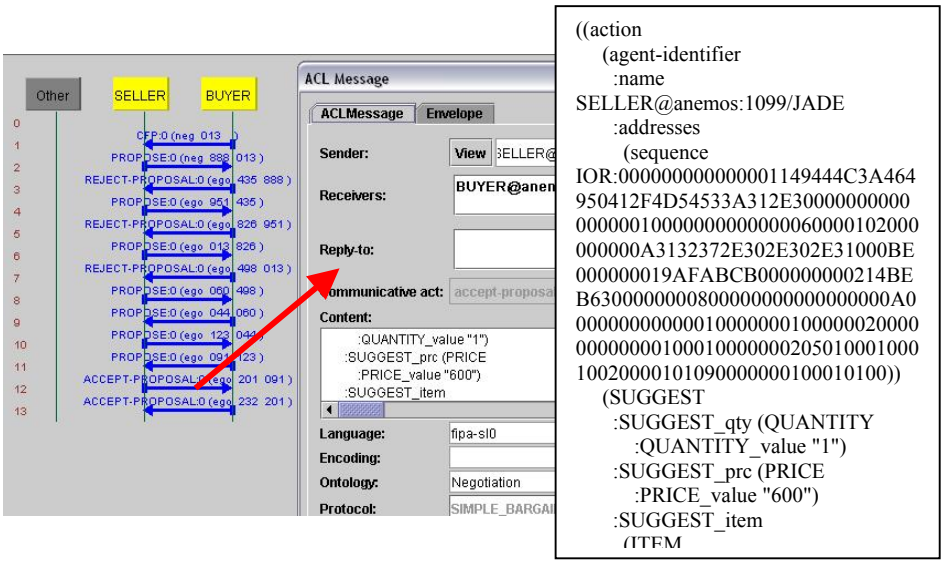


Figure 31. Negotiation ACL message 12 - Seller accepts 600

4.3 Brokering System Implementation

In this section we present an architecture for a brokering system. We identify four roles: The Buyer (or Service Requester), the Seller (or Service Provider), the Broker and the Directory Facilitator (or Yellow Pages Service). We assume that agents do not know the addresses of each other and use the yellow pages service to subscribe, or search for a particular service. The implementation concerns a hybrid technique for brokering. Offers or advertisements of products and services are expressed in RDF and users' preferences are expressed in defeasible logic. The domain of the product or service is modeled in an RDFS ontology. RDF suite [44] is used for the storing and the processing of the domain ontology. The main interaction protocol is FIPA request and is hard-coded to all the participating agents. The allowed actions an agent can perform and other attributes regarding brokering procedure itself (not product's special characteristics) are expressed in a message content ontology in protégé.

4.3.1 Brokering System Architecture

In this section, we present the architecture of the brokering system. We firstly describe the basic modules of the system and consequently the interactions among the agents, which participate in a brokering scenario. The overall architecture of the *brokering system* is depicted in Fig. 32.

4.3.1.1 Description of the Modules

RDF Translator

The role of the RDF translator is to transform the RDF statements into logical facts, and the RDFS statements into logical facts and rules. This transformation allows the RDF/S information to be processed by the rules provided by the Service Requester (representing the requester's requirements and preferences). For RDF data, the SWI-Prolog RDF parser is used to transform them into an intermediate format, representing triples as *rdf(Subject, Predicate, Object)*. Some additional processing (i) transforms the facts further into the format *Predicate(Subject, Object)*; (ii) cuts the namespaces and the "comment" elements of the RDF files, except for resources that refer to the RDF Schema, for which namespace information is retained.

In addition, for processing RDF Schema information, the following rules capturing the semantics of RDF Schema constructs are created:

A: C(X):- rdf:type(X,C).

B:C(X):- rdfs:subClassOf(Sc,C),Sc(X).

C:P(X,Y):-rdfs:subPropertyOf(Sp,P),Sp(X,Z).

D: D(X):- rdfs:domain(P,D),P(X,Z).

E: R(Z):- rdfs:range(P,R),P(X,Z).

Let us consider rule B that captures the meaning of the subclass relation of RDFS. A class Sc is subclass of a class C when all instances of Sc are also instances of C. Stated another way, if X is an instance of Sc then it is also instance of C. That is exactly what rule B says. All the above rules are created at compile-time, i.e. before the actual querying takes place. Therefore, the above rules although at first they seem second-order, because they contain variables in place of predicate names, they are actually first-order rules, i.e. predicate names are constant at run-time.

Semantic-Syntactic Validator

This module is an embedded version of [81], a parser for validating RDF descriptions. Upon receipt of an advertisement, the RDF description, which corresponds to that advertisement, is checked by this module. Among others, the tests that are being performed are: class hierarchy loops, property hierarchy loops, domain/range of subproperties, source/target resources of properties and types of resources. This module is "called" by the control module and returns either the RDF description, in case the latter is error free, or an error message. For the implementation of this module we used the API of VRP.

Interaction and Communication Modules

The communication module is responsible for sensing the network and notifying the control module when an external event (e.g. a request message) occurs. In order to decide the course of action based on the incoming message's type, the broker agent extracts the message from the queue and examines its type, i.e. whether it is a "Broker Request", "Advertise Request" message etc. Accordingly it activates the interaction module. Interaction module consists of different interaction protocols that extend the standard FIPA Request interaction protocol. For the implementation of these modules, we used the API of JADE framework.

RDF Suite Module

The RDF Suite module is responsible for all the actions related with the handling of the advertisements and the domain ontology. The most important functions of this module are:

- Initial storing of RDFS ontology and RDF instances into RDF repository
- Update of RDF repository with RDF descriptions that are sent by the service providers and correspond to product or service advertisements.
- Preparation of an RQL query and forwarding to RDF suite.
- Acquisition of RQL query's results

The RDF suite module implements a client socket, which connects to a server socket and passes an RQL query or retrieves the results. At this point we must say, that although RSSDB and VRP work well in windows and their Java API can be easily used, there is a problem with RQL that operates only in UNIX. The server socket, which creates a UNIX pipe to RSSDB, solves this problem. For the implementation of this module we used the API of RSSDB and the API of java for File Management and Networking.

Rule-Query-RDF Loader

The role of this module is to download the files, which contain the rules and the query of the user, in defeasible logic format. It also loads the appropriate RDF data, which correspond to service provider's advertisements. It also implements methods for file handling. For the implementation of this module we used the API of java for File Management and the API for Networking.

Reasoning Module

The role of the Evaluator is to apply the queries to files, which contain the facts and the rules, and to evaluate the answer. When the Service Requester makes a query, the Evaluator compiles the files containing the facts and the rules, and applies the query to the compiled files. The answer of the query is sent to the Control Module of the system. XSB Prolog is used as the compiler and evaluator for our system. We made this choice, as we needed a logic programming system supporting well-founded semantics. XSB Prolog offers this functionality through its *tnot* operator.

4.3.1.2 Description of the Interactions

We describe the sequence of actions, separately for the buyer (red colour) and the seller (blue colour). Initially Broker agent, subscribes to Directory Facilitator or D.F. agent and Buyer and Seller search for a brokering service. These actions are depicted by the orange dashed lines (step 0). They provide information such as the ontologies they are committed to, the interaction protocols they use, the content language they use etc.

A *seller* initializes an interaction by issuing an "Advertise" request (step 1). The broker extracts the field "RDFInfoAtWeb" from the received message and tries to download the corresponding advertisement from the web, which is an RDF description with information about the advertised product or service (step 2). If the document exists, broker informs the seller that he agrees to perform the requested action (step 3). Subsequently the broker checks the RDF advertisement semantically and syntactically, using the Semantic and Syntactic validator module (step 4). The result is returned to the control module of broker (step 5), and if the advertisement is valid, according to the well-known domain ontology, seller is informed that the requested action was performed. Otherwise an error message is posted (step 6). Broker then performs a twofold action. He firstly feeds the RDF suite module with the advertisement (step 7), which in turn stores it to the RDF suite repository (step 8) and secondly sends the advertisement to the RDF translator module of the Inference engine to add it in the subsumption hierarchy (step 9). Finally the knowledge base is updated with the new facts, which were previously extracted from the RDF advertisement (step10).

A *buyer* sends an "Available Products" request to the broker (step 1). Broker informs buyer if he agrees or not to perform the action (step 2) and in turn he sends to buyer a message with the available categories of products (step 3). Buyer then issues a "Brokering" request, for a particular category of products (step 4). Broker downloads from the web the rules, which capture the preferences of the buyer and are expressed in defeasible logic. He also downloads the submitted query, which is also expressed in defeasible logic (step 5). Subsequently the rules and the query are stored in the knowledge base by the broker (step 6). Broker informs the buyer if he agrees or not to perform the requested action, according to the validity of the rules and the query (step 7). The check is performed by the Reasoning module but is out of the scope of this thesis. Broker in turn activates the reasoning module (step 8), which uses the stored data in the knowledge base (step 9) and performs the reasoning process. The result is all the unique codes, which practically are the ID's of the RDF descriptions (step 10).

Afterwards RDF Suite module of broker creates dynamically an RQL query (step 11) that is passed to the RDF suite repository for the retrieval of all the resources, which correspond to the previously resulted ID's (step 12). The result of the query is returned to the broker (step 13). Lastly broker encapsulates the received information to an ACL message and sends it to the buyer (step 14).

4.3.2 Message Content Ontology for Brokering

We used an ontology particularly for actions, which agents can request and perform during the brokering procedure and predicates that are error messages. The most important agent actions of our M.C.O are: ADVERTISE, BROKERREQUEST and AVAILPRODUCTREQUEST. We used Protégé for the design of the ontology and Beangenerator, a tool that transforms Ontologies defined in protégé model or RDFS into java classes, for easiest manipulation by the agents. M.C.O can be viewed in detail in Fig.33.

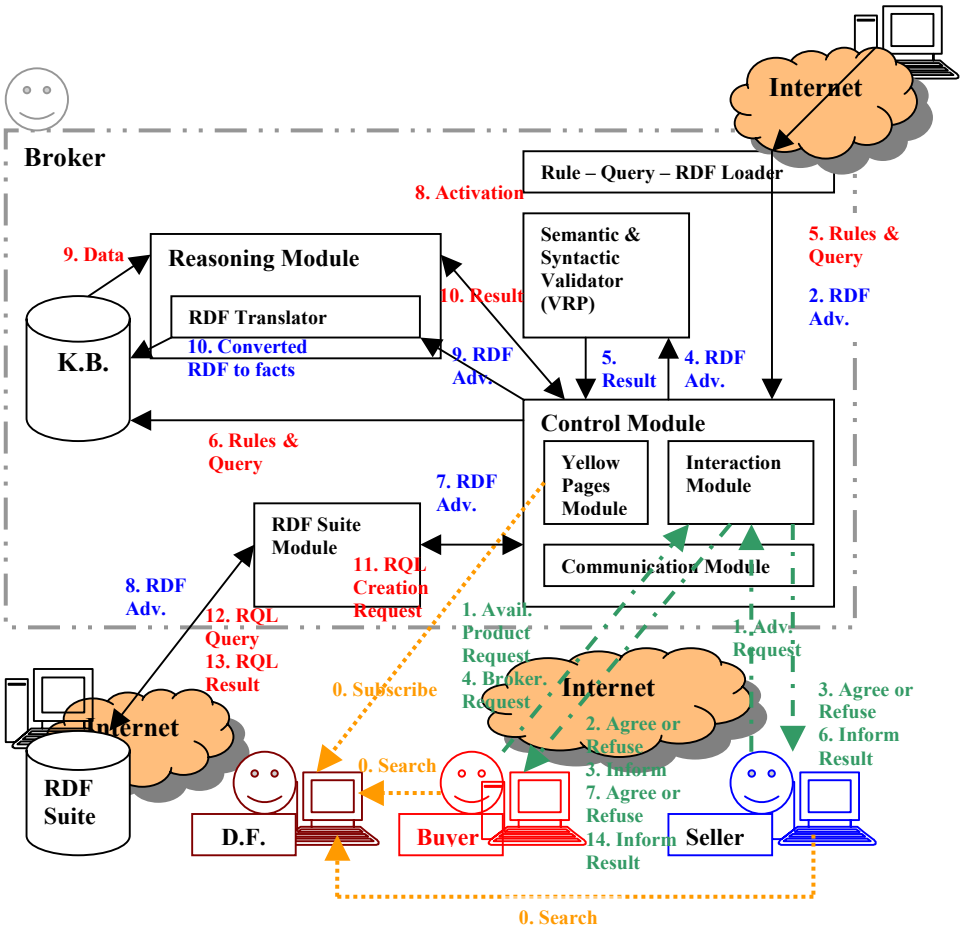


Figure 32. Brokering System Architecture

Figure 33. Brokering Message Content Ontology

4.3.3 FIPA Request Interaction Protocol

As all the agents of our brokering system use FIPA Request Interaction Protocol, we provide some information about its basic functionality.

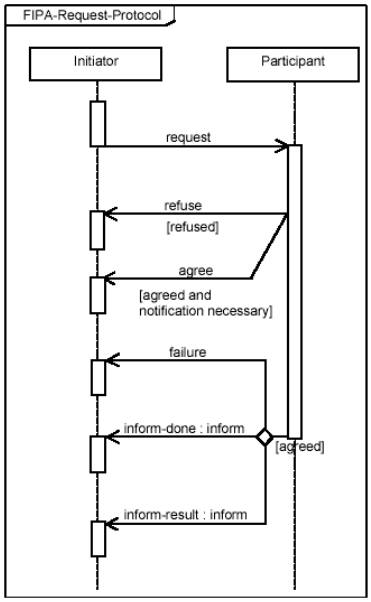


Figure 34. Fipa Request Interaction Protocol

The FIPA Request Interaction Protocol allows one agent to request another to perform some action. The Participant processes the request and makes a decision whether to accept or refuse the request. If a refuse decision is made, then “refused” becomes true and the Participant communicates a refuse message. Otherwise, “agreed”

becomes true. If conditions indicate that an explicit agreement is required (that is, "notification necessary" is true), then the Participant communicates an agree. The agree may be optional depending on circumstances, for example, if the requested action is very quick and can happen before a time specified in the reply-by parameter. Once the request has been agreed upon, then the Participant must communicate either:

- A failure if it fails in its attempt to fill the request,
- An inform-done if it successfully completes the request and only wishes to indicate that it is done, or,
- An inform-result if it wishes to indicate both that it is done and notify the initiator of the results.

Any interaction using this interaction protocol is identified by a globally unique, non-null conversation-id parameter, assigned by the Initiator. The agents involved in the interaction must tag all of its ACL messages with this conversation identifier. This enables each agent to manage its communication strategies and activities; for example, it allows an agent to identify individual conversations and to reason across historical records of conversations.

JADE provides libraries that encode the behavior of FIPA Request Interaction Protocol.

4.3.4 A Brokering Scenario

As we have presented the architecture of our brokering system and examined its basic characteristics, we make a scenario to get a better insight into system's operation. We choose to model the domain of tourism industry. We define an ontology in RDFS to model all the concepts and their relationships in the above domain. We also define a few initial instances of the ontology regarding descriptions about hotels, islands, travel agents, means of transport, car rental companies etc. The ontology and the initial instances can be found in Appendix A.

4.3.4.1 Expression of Offers

Every travel agent company, which is a potential service provider, can publish an offer to the broker. After the publication, the offer is considered as an advertisement. The advertisement regards a complete travel package and its format is depicted in Fig. 35. This advertisement is submitted by Zorpidis S.A. and regards a package for two persons for the island of Crete. It includes a hotel, local transportation service (a car) and tickets for the ferry, and costs 1000 €. For our scenario we assume that three more

travel agent companies publish an advertisement. The RDF description, which corresponds to their offers, can also be found in Appendix A.

```
<?xml version="1.0" ?>
<!DOCTYPE rdf:RDF (View Source for full doctype...)>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:instances="http://www.csd.uoc.gr/~dogjohn/data/TravelOntologyInstances.rdf#"
 xmlns:schema="http://www.csd.uoc.gr/~dogjohn/ontology/TravelOntology.rdfs#">
<schema:Itinerary rdf:ID="IT1" />
<rdf:Description rdf:about="#IT1">
 <rdf:type
 rdf:resource="http://www.csd.uoc.gr/~dogjohn/ontology/TravelOntology.rdfs#Itinerary"/>
 <schema:from rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
   thessaloniki
 </schema:from>
 <schema:to rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
   Creta
 </schema:to>
 <schema:departureDate rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
   3/8/2004
 </schema:departureDate>
 <schema:returnDate rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
  15/8/2004
 </schema:returnDate>
 <schema:persons rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
   2
 </schema:persons>
 <schema:price rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
   1000
 </schema:price>
 <schema:offeredBy
 rdf:resource="http://www.csd.uoc.gr/~dogjohn/data/TravelOntologyInstances.rdf#ZORP" />
 <schema:includesResort
 rdf:resource="http://www.csd.uoc.gr/~dogjohn/data/TravelOntologyInstances.rdf#CretaMareRoyal" />
  <schema:includesTransportation
 rdf:resource="http://www.csd.uoc.gr/~dogjohn/data/TravelOntologyInstances.rdf#Minoan" />
 <schema:includesService
 rdf:resource="http://www.csd.uoc.gr/~dogjohn/data/TravelOntologyInstances.rdf#AVIS" />
 <schema:forPlace
 rdf:resource="http://www.csd.uoc.gr/~dogjohn/data/TravelOntologyInstances.rdf#Creta" />
 </rdf:Description>
</rdf:RDF>
```

Figure 35. Expression of an Advertisement in RDF

4.3.4.2 Formalization of Requirements and Preferences

On the other hand, every customer who is a potential service requester can express his requirements and preferences to the broker. Consider the following example: “Antonis, a busy businessman, has the following preferences about his holiday travel package: First of all, he wants to depart from Athens and he considers that the hotel at

the place of vocation must offer breakfast. In addition, he would like either the existence of a swimming pool at the hotel to relax all the day, or a car equipped with A/C, to make daily excursions at the island. The swimming pool is more important to him than the car. However, Antonis believes that if there is no parking area at the hotel, the car is useless, because he adds to him extra effort and fatigue. Lastly, if the tickets for his transportation to the island, are not included in the travel package, he is not willing to accept it…". This verbal description of Antoni's preferences can be modeled through the following defeasible logic rules, depicted in Fig. 36.

```
r1: from(X,athens),includesResort(X,Y),breakfast(Y,true)
⇒ accept(X)
r2: from(X,athens),includesResort(X,Y),swimmingPool(Y,true)
⇒ accept(X)
r3: from(X,athens), includesService(X,Z), hasVehicle(Z,W),
vehicleAC(W,true) ⇒ accept(X)
r4: includesResort(X,Y),parking(Y,false) ⇒ ~accept(X)
r5: ~includesTransportation(X,Z) ⇒ ~accept(X)

    r4 > r3, r5 > r1, r5 > r2, r5 > r3, r2 > r3
```

Figure 36. Expression of User's Requirements in Defeasible Logic

4.3.5 Brokering Trace

Based on the above scenario, we present a trace of the brokering procedure. We consider that seller1, seller2, seller3 and seller4 advertise the travel packages with codes IT1, IT2, IT3 and IT4 respectively (Appendix A). We also consider that buyers: buyer1, buyer2 and buyer3 are in fact one buyer who requests a brokering service during different time periods. We assume that in the beginning, there are no facts and no RDF descriptions regarding advertisements. There are only facts regarding the initial data and the subsumption hierarchy of the ontology. Initially broker registers himself to the directory facilitator as depicted in Fig. 37 and the latter sends him am acknowledgement message.(REQUEST:0 , INFORM:0). Seller 2 in turn searches the directory facilitator for the broker and when he learns about his address, he sends him an "Advertise" message. This message is depicted in Fig.37. The field ITEM_rdfInfoAtWeb, informs broker for the location of the actual advertisement in RDF form.

Figure 37. Brokering Trace 1

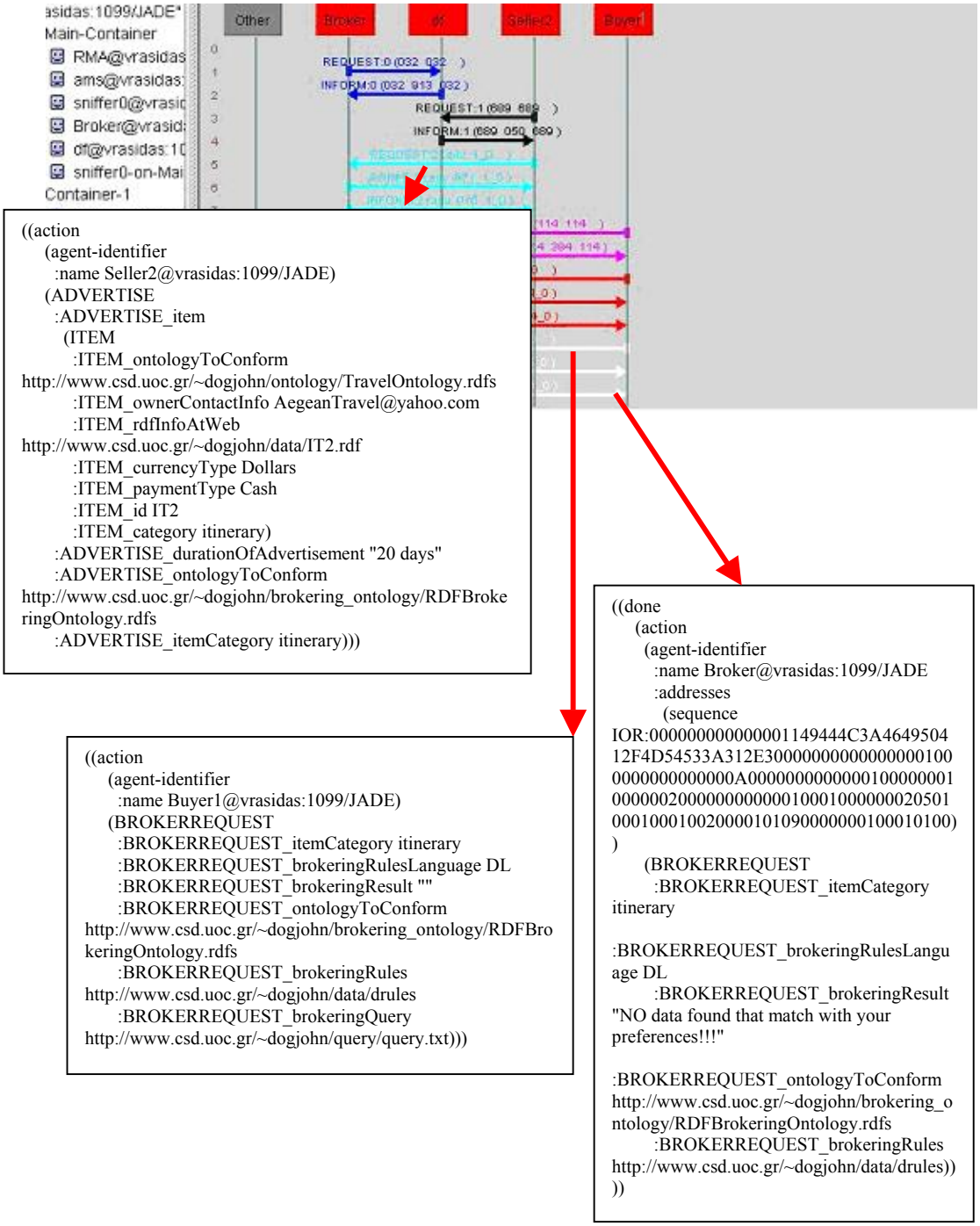
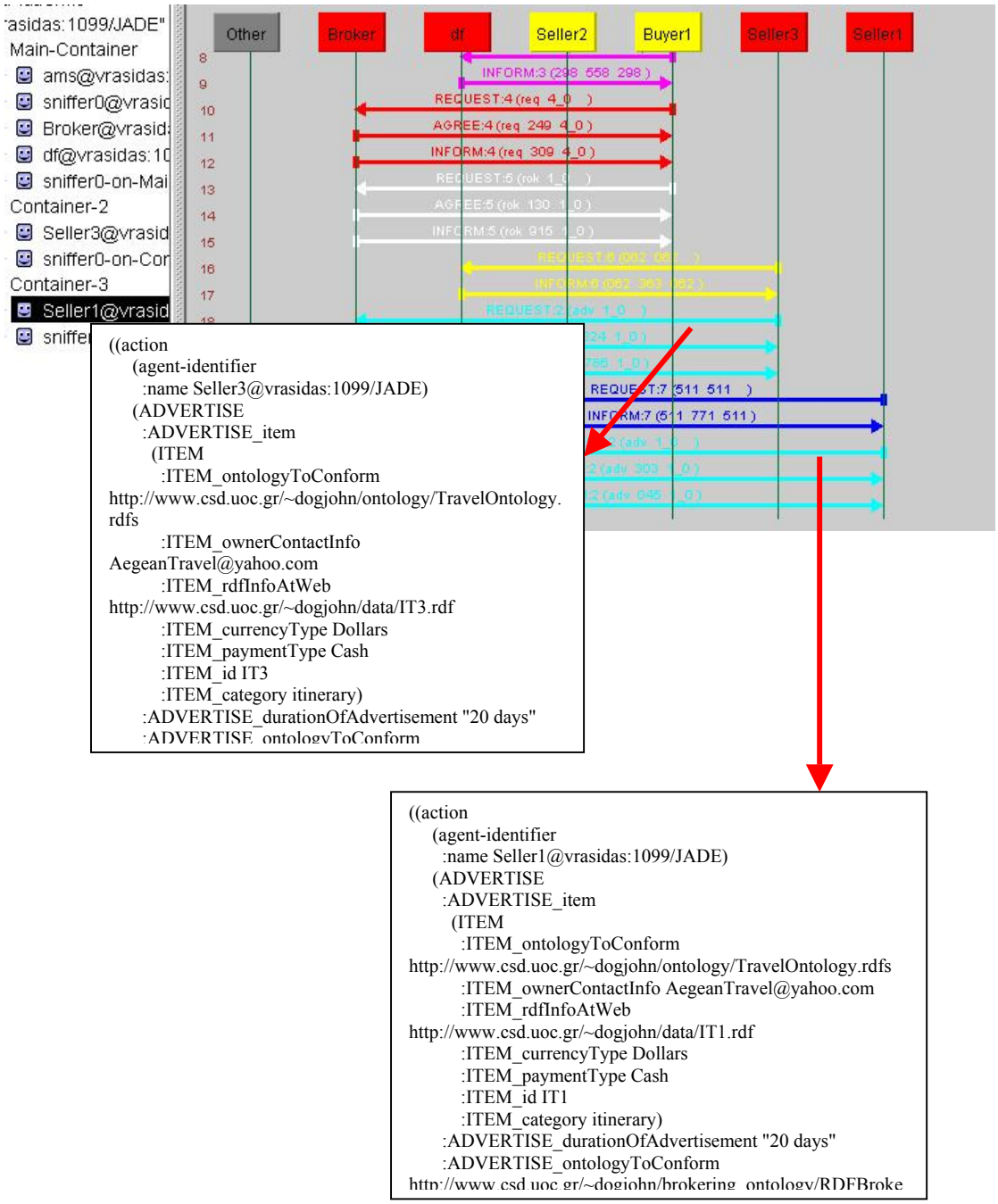


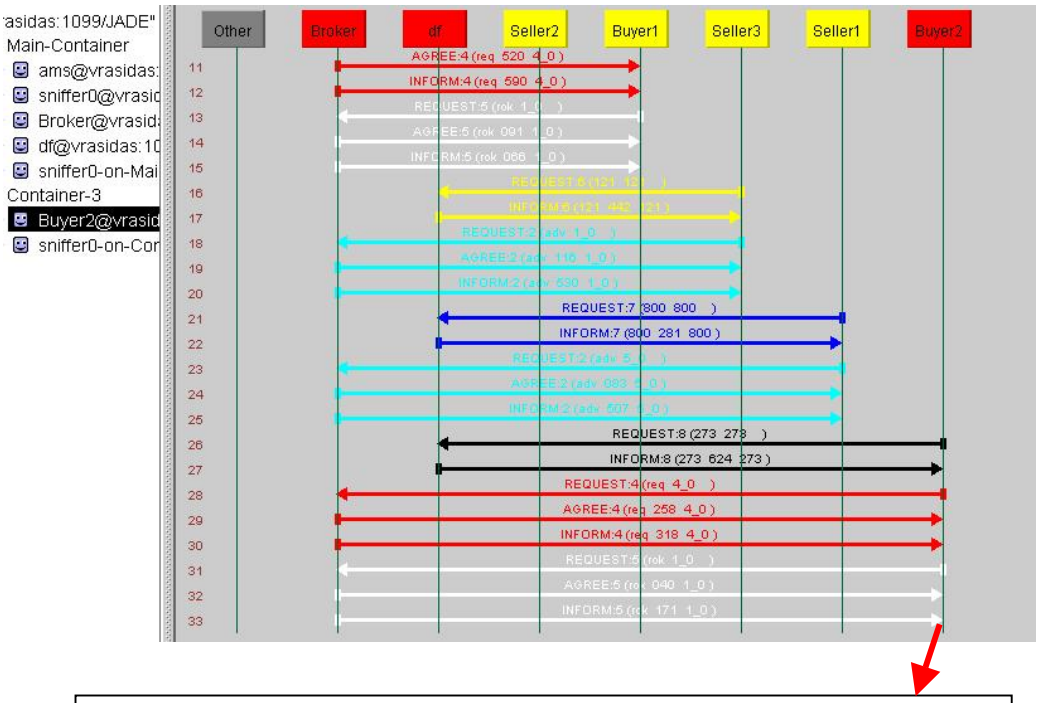
Figure 38. Brokering Trace 2



Buyer 1 consequently issues a “Brokering” request (Fig.37). In this message buyer provides to the broker information such as, the URL address of the rules and the query along with the brokering category he is interested in and the language he uses for brokering. The “answer” of the broker informs buyer that at this time there are no data that match with his preferences (Fig. 37). After a while, two more sellers, named seller 3 and seller 1 advertise their travel packages to the broker (Fig. 38). As the

advertisement IT3 of seller 3 matches with buyer's preferences the next time the latter issues a broker request he receives a message that contains the relevant resource. (Fig. 39) Lastly, seller 4 advertises his travel package and when buyer 3 requests for brokering, broker sends him a message that contains information about the relevant resources, including IT4 (Fig. 40).

Figure 39. Brokering Trace 3




```
((done
    (action
      (agent-identifier
        :name Broker@vrasidas:1099/JADE
        :addresses
          (sequence
IOR:000000000000001149444C3A464950412F4D54533A312E3000000000000000010000000000000
0640001020000000E3133392E39312E3138332E343000062E00000019AFABCB0000000002234B9
5170000000800000000000000000A0000000000001000000010000002000000000000100010000000 2
0501000100010020000101090000000100010100))
      (BROKERREQUEST
        :BROKERREQUEST_itemCategory itinerary
        :BROKERREQUEST_brokeringRulesLanguage DL
        :BROKERREQUEST_brokeringResult "
<?xml version="1.0" encoding="UTF-8"?>

<RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

<rdf:Bag>
<rdf:li rdf:type="resource" rdf:resource="http://www.csd.uoc.gr/~dogjohn/data/IT3.rdf#IT3"/>
</rdf:Bag>
</RDF>

"
        :BROKERREQUEST_ontologyToConform
http://www.csd.uoc.gr/~dogjohn/brokering_ontology/RDFBrokeringOntology.rdfs
        :BROKERREQUEST_brokeringRules http://www.csd.uoc.gr/~dogjohn/data/drules))))
```

```
((action
    (agent-identifier
      :name Seller4@vrasidas:1099/JADE)
    (ADVERTISE
      :ADVERTISE_item
       (ITEM
         :ITEM_ontologyToConform http://www.csd.uoc.gr/~dogjohn/ontology/TravelOntology.rdfs
         :ITEM_ownerContactInfo AegeanTravel@yahoo.com
         :ITEM_rdfInfoAtWeb http://www.csd.uoc.gr/~dogjohn/data/IT4.rdf
         :ITEM_currencyType Dollars
         :ITEM_paymentType Cash
         :ITEM_id IT4
         :ITEM_category itinerary)
      :ADVERTISE_durationOfAdvertisement "20 days"
      :ADVERTISE_ontologyToConform
http://www.csd.uoc.gr/~dogjohn/brokering_ontology/RDFBrokeringOntology.rdfs
```

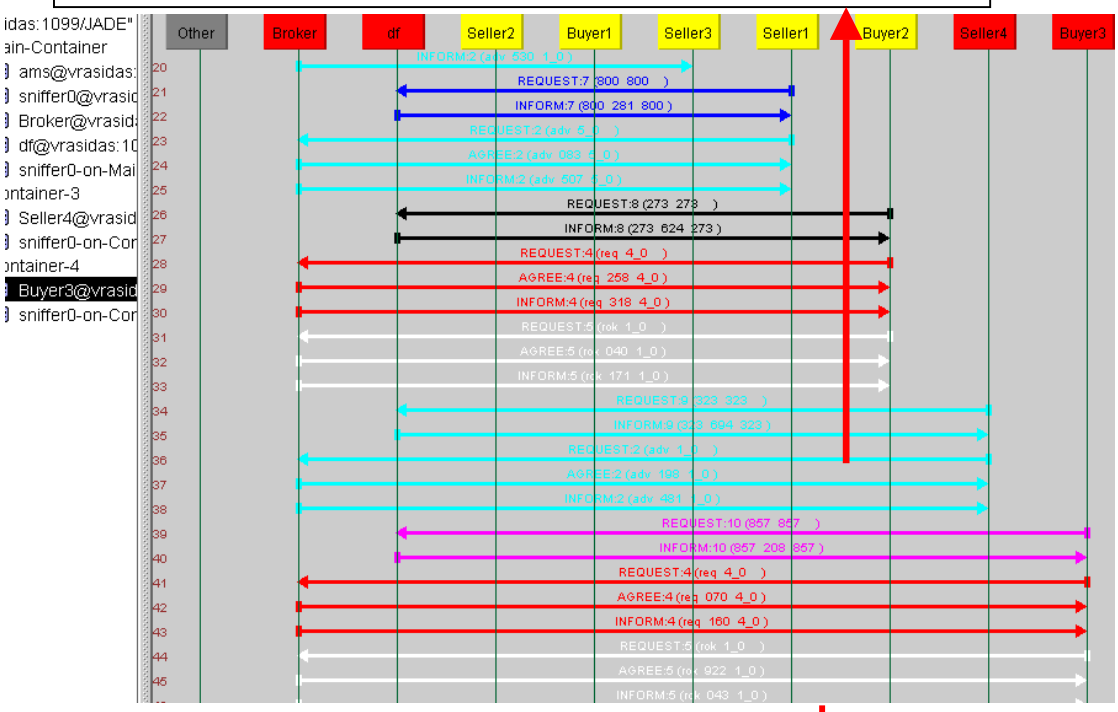


Figure 40. Brokering Trace 4

```
((done
    (action
      (agent-identifier
        :name Broker@vrasidas:1099/JADE
        :addresses
         (sequence
IOR:000000000000001149444C3A464950412F4D54533A312E3000000000000000010000000000000064000102000000000E3133392E39312E3138332E343000062E00000019AFABCB0000000002234B95170000000800000000000000000A000000000000010000001000000200000000000001000100000002050100010001002000010109000000100010100))
      (BROKERREQUEST
        :BROKERREQUEST_itemCategory itinerary
        :BROKERREQUEST_brokeringRulesLanguage DL
        :BROKERREQUEST_brokeringResult "
<?xml version="1.0" encoding="UTF-8"?>

<RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

<rdf:Bag>
<rdf:li rdf:type="resource" rdf:resource="http://www.csd.uoc.gr/~dogjohn/data/IT3.rdf#IT3"/>
<rdf:li rdf:type="resource" rdf:resource="http://www.csd.uoc.gr/~dogjohn/data/IT4.rdf#IT4"/>
</rdf:Bag>
</RDF>
```

4.4 Implementation Choices and Conventions – System Limitations

4.4.1 Choice of Inference Engines

For the reasoning module of the negotiating agent we used [56]. The system is called DR-DEVICE and is capable of reasoning about RDF metadata over multiple Web sources using defeasible logic rules. The system is implemented on top of CLIPS production rule system and builds upon DR-DEVICE, an earlier deductive rule system over RDF metadata that also supports derived attribute and aggregate attribute rules. Rules can be expressed either in a native CLIPS-like language, or in an extension of the OO-RuleML syntax. We chose the particular system because it employs forward-chaining technique, something that is compliant with the nature of a negotiation process. That means that, as the strategy of the agent is expressed through rules, the most appropriate rule must fire according to the input data.

For the reasoning module of the broker we used [54], a system developed at FORTH. It is a Prolog-based implementation of defeasible logic, and accordingly employs a backward chaining technique. The latter feature seems to fit with the nature of brokering. In this case we want to know if a rule fires or not and we submit an explicit query for that purpose.

4.4.2 Choice of FIPA ACL Content Language

The only “standard” ACL content language proposed by FIPA up to date is SL. However, working groups of FIPA also experiment with KIF, CCL and RDF. In JADE an SL and an RDF Codec are available. They transform the content of the message from SL and RDF to java classes and reversely, for easiest manipulation by the agent. However, these Codecs have a serious drawback. They support the use of *only one* ontology and consequently one namespace, defined by the user. We chose SL Codec for our implementation because RDF Codec seems to have additional problems. The problems of RDF Codec become apparent, if we examine again the message of Fig. 20. Firstly, the agent action SUGGEST, that we use in our ontology appear to be of <fipa-rdf:type>SUGGEST</fipa-rdf:type> while it should be for example of type <Negotiation:SUGGEST>, as the action SUGGEST is defined in our ontology. The problem comes with the Codec. The only type the current implementation allows to define is <fipa-rdf:type>. Secondly, there is a problem with the agent identifier. While the SL content language specification (line 685) states that “An agent is represented by its agent-identifier using its standard format”, there is no such statement in FIPA RDF

content language specification. This ubiquity is also reflected to the ACL message of Fig. 20. The name of the agent seems to be <Negotiation:name> BUYER@anemos:10999/JADE </Negotiation: name> while for example it should be of type <fipa-rdf:name>. These limitations in conjunction with the support of a single namespace discouraged us from using RDF Codec, which is open for many improvements.

Another decision we took was that the RDF description of an advertisement is downloaded by the broker from seller's web site. We could alternatively wrap the RDF description in a field inside the content of an ACL message. We find no important difference between these two approaches. However, since we choose SL as a content language we completely separate the description of the advertisement that is in RDF format.

An interesting approach regarding content language is adopted in [39]. Zou et al., use OWL as a content language for agent communication. OWL has a well-defined model-theoretic semantics. Its expressive power is indisputable and, of course, offers better support for using terms drawn from multiple ontologies, than the previously mentioned popular ACL content languages. They define the FIPA-OWL ontology for use as a FIPA-compliant content language. In addition to the basic required classes (Agent ACLMessage, Service) and necessary expressive requirements (such as Propositions Actions and Reification) they provide support for rules and queries. They use FOWL (Flora OWL) to represent and reason about content, presented in OWL. On receiving a message an agent parses its content into triples, which are then loaded into the XSB engine for processing. However, as we wanted to implemented only simple actions regarding brokering in a closed environment, we found SL Codec satisfactory. In addition SL Codec seems to be faster something that is useful in our case, because the translation of RDF into logical facts and the inference process are intrinsically time-consuming. So we wanted to avoid the extra overhead of parsing and reason about the content of messages written in RDF or OWL.

4.4.3 Choice of Web Ontology Language

We have chosen RDF over the use of OWL, because at present it is not clear how the deductive capabilities of OWL and rule systems can be combined; it is one of the main research lines in the Semantic Web community. We could certainly use most features of OWL Lite, given that they can be expressed using rules.

4.4.4 System Limitations

The most important limitations of our systems are the following:

- The syntax of DR-Device is too complicated.
- When it comes to brokering, only the addition of new facts is possible and not the removal. (When an advertisement expires, it must be retracted).
- The inference engine of brokering system does not support arithmetic capabilities.

5. Conclusions and Future Work

5.1 Conclusions and Discussion

So far, we have presented the implementation of a negotiating agent architecture and a semantic brokering system, using defeasible logic, and exploiting semantic web and agent-based technologies. The combination of these technologies seems to be fascinating and challenging. During the research and the development phase of this thesis a lot of conclusions were drawn. Some of them are more general while others are more concrete and regard our implementation. We analyze them in the following paragraphs.

Firstly, defeasible logic seems to be a very promising solution when it comes to negotiation strategies and brokering preferences modeling. It combines all the desired characteristics of a language for data modeling. More specific, it is formal with well-defined semantics, conceptual with a high potential of abstraction, comprehensive, modular and executable. It is also highly expressive as it can model a great number of diverse cases as we have seen in section 3.4. However, there are a lot of remaining open issues. The current implementations of defeasible logic [54], [56] are still in an experimental state of research prototypes. In addition, both above implementation use platform dependent syntax for the expression of rules, either clips-like or prolog-like, something that makes the rules complicated. As a conclusion there is a need for writing rules in a more abstract syntax and for additional capabilities such as a constraint solver, which could be of great importance in some types of negotiation such as multi-attribute negotiation.

Secondly, the relationship and potential interoperability of agents and semantic web technology is under question. Indicative is the title of [52]: "The Semantic Web and Software Agents: Partners or Just Neighbours". The same article enumerates potential combined uses of agents and semantic web but also stresses existing problems. It argues that as agent standards, such as FIPA, matured before the widespread use of RDF and similar technologies a number of incompatibilities have occurred. For example, symbols in the Semantic Web are URI's, either written out in full or abbreviated in q-name form (i.e., prefix:name). URIs are not strings, but must be written as strings in FIPA ACL for consistency with the ACL grammar. Some performatives are missing (such as query from FIPA specification). In addition on the

Semantic Web, URL typically references ontologies, but a given fragment of OWL may reference a number of ontologies by name. This makes the use of the :ontology message parameter unclear. None of these problems are insurmountable, but if every agent project adopts different conventions, interoperability is reduced.

Thirdly, as the use of RDF and OWL start to be of increasing popularity as FIPA ACL content languages, JADE development board should seek the improvement of RDF Codec and may develop an OWL Codec in addition to already existing SL, XML and RDF ones. On the other hand many approaches start to use tools such as Jena for the processing of the exchanged messages, as an alternative to the mentioned Codecs. Thus, FIPA and JADE communities must publish general accepted ontologies in RDF or OWL that will encode all the basic concepts for a message content language (Agent ACLMessage, Service) along with other necessary expressive requirements (such as Propositions Actions and Reification) and finally support for queries. All these issues are till now open to the developer. As a result everyone develops his own ontology, which leads to lack of general consensus, incomplete approaches etc.

Fourthly, our semantic brokering implementation, based on hybrid techniques, seems to be slightly or at least the same expressive as the other existing approaches. Promising seems to be the use of RQL. However, its inability to express soft preferences is a problem. A preference elicitation mechanism should be added to the existing functionality. (Preference-RQL may be?).

Fifthly, we have described in section 3 a multi-attribute negotiation scenario. In our current negotiation implementation the only negotiation attribute is price. As many attributes or many products are subject to negotiation, the need for the existence of powerful domain ontology in RDFS or OWL is indisputable.

Finally, during the negotiating agent development, we initially experimented with the expression of negotiation strategies in Java. Although the reasoning process was considerably faster, we found that the results were not always explainable. In other words the logical analysis of the agent's decisions was impossible. In contrary defeasible logic makes the logical analysis a much easier task. Lastly, the expression of the strategy with rules helped as to find errors in the negotiation protocol that initially (when the strategy was expressed in Java) went unnoticed.

5.2 Future Work

In the future we plan to:

- Integrate the currently separate systems for negotiation and brokering. In [43], an agent-based architecture for brokering and negotiation is presented. In our current implementation, a service requester agent is able to find potential products or services and potential service providers. We intend to extend our system to support direct involvement of the service requester to negotiation with the service provider for the resulted product, as soon as the brokering stage has been completed.
- Implement graphical user interfaces for the integrated system. Someone will be able to load, the files, which correspond to the rules of negotiation strategy and brokering preferences respectively, using a file manager. He will be also able to adjust negotiation protocol parameters and monitor the progress of the brokering and negotiation procedure.
- Adopt a more flexible solution for FIPA ACL content language, due to the limitation of current solutions. We intend to use an ontology language such as OWL or use RDF. We also intend to create ontology for the content language according to the standard of FIPA and an ontology for making queries. The existence of a powerful inference engine is indisputably of great importance, towards the realization of the latter solution. The context of the query will be parsed and transformed to triples along with the ontologies. These will feed the inference engine, which will combine these facts with message-content specific rules and give a result, regarding the meaning of the received message.
- Add advertisement removal functionality, which will be activated, when the advertisement has expired.
- Implement the protocols both for negotiation and brokering in defeasible logic.

6. References

e-Commerce

[1]. Minghua He, Nicholas R. Jennings, Ho-Fung Leung (2003). "On Agent-Mediated Electronic Commerce". IEEE Transactions on Knowledge and Data Engineering Vol. 15, No 4 July/August 2003.

[2]. Pattie Maes, Robert H.Guttman, Alexandros G. Moukas (1999). "Agents that Buy and Sell". Communications of the ACM Vol. 4 March 1999.

Negotiation

[3]. Agorics Inc (1996). "A survey of auctions" http://www.agorics.com/new.html, 1996.

[4]. Patricia Anthony, Wendy Hall, Viet Dung Dang, Nicholas R. Jennings (2001). "Autonomous Bidding Agents for Participating in Multiple On-line Auctions". In Proceedings of the IJCAI-2001 Workshop on E-Business and the Intelligent Web Seattle, WA.

[5]. Claudio Bartolini, Chreis Priest. (2001). "A Framework for Automated Negotiation". HP Technical Report: HPL-2001-91.

[6]. Claudio Bartolini, Chris Preist and Nicholas R. Jennings (2002). "A Generic Software Framework for Automated Negotiation". In Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS) Bologna Italy 2002.

[7]. Martin Bitchler and Jayant Kalagnanam (2002). "Bidding Languages and Winner Determination in Multi-Attribute Auctions". IBM Research Report RC 22478.

[8]. A. Chavez, D. Dreilinger, R. Guttman, and P. Maes (1997). "A Real-life Experiment in Creating an Agent Marketplace". In Proceedings. of the 2nd Int. Conference on The Practical Applications of Agents and Multi-Agent Technology (PAAM), London, UK, 1997.

[9]. Eva Chen, Gregory Kersten, Rustam Vahidov (2004). "An e-marketplace for Agent Supported Commerce Negotiations". Appeared in 25th McMaster World Congress Management of Electronic Business January 2004.

[10]. Marlon Dumas, Guido Governatori, Arthur H.M ter Hofstede, Phillipa Oaks (2002). "A Formal Approach to Negotiating Agents Development". Elsevier Science – Electronic Commerce Research and Applications Vol. 1,Issue 2 Summer 2002 pp. 193-207.

[11]. Guido Governatori, Marlon Dumas, Arthur H.M ter Hofstede, Philipa Oaks (2001). "A Formal Approach to Protocols and Strategies for Legal Negotiation". In Proceedings of the 8th

International Conference on Artificial Intelligence and Law St. Louis, Missouri, United States pp. 168 - 177.

[12]. B. Grosof, Terrence Poon (2003) "SweetDeal: Representing Agent Contracts with Exceptions using XML Rules, Ontologies, and Process Descriptions". In Proceedings of the 12th International Conference on WWW. Budapest Hungary 2003. pp. 340- 349.

[13]. Robert H. Guttmann, Pattie Maes (1998). Cooperative vs. Competitive Multi-Agent Negotiations in Retail Electronic Commerce. In Proceedings of the 2nd International Workshop on Cooperative Information Agents II, Learning, Mobility and Electronic Commerce for Information Discovery on the Internet. pp. 135-147.

[14]. N.R.Jennings, S.Parsons, C.Sierra, P.Faratin (2000). "Automated Negotiation ". In Proceedings of 5th International. Conference on the Practical Application of Intelligent Agents and Multi- Agent Systems (PAAM-2000).

[15]. Iyad Rahwan, Liz Sonemberg, Franc Dignum (2003). "Towards Interest-Based Negotiation". In Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi Agents Systems (AAMAS) Melbourne Australia 2003.

[16]. H. Raiffa (1982). "The Art and Science of Negotiation". Harvard University Press.

[17]. Daniel M. Reeves, Michael P. Wellman, Benjamin N.Grosof, Hoi Y. Chan (2000). "Automated Negotiation from Declarative Contract Descriptions". In Proceedings of 17th National Conference on Artificial Intelligence, Workshop on Knowledge-Based Electronic Markets (KBEM), Austin, Texas, July 30–31 2000.

[18]. David Lucking-Reiley (2000). "Auctions on the Internet. What's Being Auctioned and How". Journal of Industrial Economics 48. August 2000 pp.227-252.

[19]. J. S. Rosenschein and Gilad Zlotkin (1994). "Rules of Encounter: Designing Conventions for Automated Negotiation among Computers". MIT Press, Readings MA, USA, 1994.

[20]. Tuomas SandHolm (2002). "eMediator: A Next Generation Electronic Commerce Server". Journal of Computational Intelligence Vol.18, No. 4 2002.

[21]. Carles Sierra, Nick R. Jennings, Pablo Noriega, Simon Parsons (1997). "A Framework for Argumentation-based Negotiation". In Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures and Languages. pp 177-192.

[22]. Stanley Y. W. Su, Chunbo Huang and Joachim Hammer (2000). "A Replicable Web-based Negotiation Server for E-Commerce". In proceedings of the 33rd Hawaii International Conference on System Sciences. 2000.

[23]. Edward Tsang and Tim Gosling (2002). "Simple Constrained Bargaining Game". In proceedings of the Distributed Constrained Satisfaction Workshop- 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS) Bologna Italy 2002.

[24]. Peter R. Wurman, Michael P, Wellman William E, Walsh (1998). "The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents". In Proceedings of the 2nd International Conference on Autonomous Agents. Minneapolis, Minnesota. pp. 301-308.

[25]. Ian Dickinson, Michael Wooldridge (2003). "Towards Practical Reasoning Agents for the Semantic Web". In Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS) Melbourne Australia 2003.

[26]. Stanley Y.W.Su, Chunbo Huang, Joachim Hammer, Yihua Huang, Haifei Li, Liu Wang, Youzhong Liu, Charnyote Pluempitiwiriyawej, Minsoo Lee, Herman Lam (2001) " An Internet-based Negotiation Server for e-Commerce". The VLDB Journal 10. pp 72-90.

Semantic Brokering

[27]. Y-C. Chen, W-T. Hsu, P-H. Hung (2003). "Towards Web Automation by Integrating Semantic Web and Web Services". In Proceedings of the. 12th International Conference on WWW. Budapest Hungary 2003.

[28]. L. Li and I. Horrocks (2003). "A Software Framework for Matchmaking Based on Semantic Web Technology". In Proceedings of the 12th International Conference on WWW. Budapest Hungary 2003.

[29]. M. Nodine, W. Bohrer, A. Hee Hiong Ngu (1998). "Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth". In Proceedings of the 15th International Conference on Data Engineering, IEEE Computer Society.

[30]. M. Paolucci, T. Kawamura, T.R. Payne, K. Sycara (2002). "Semantic Matching of Web Services Capabilities". In Proceedings of the 1st International Semantic Web Conference (ISWC-2002).

[31]. D. Trastour, C. Bartolini, J. Gonzalez- Castillo (2001). "A Semantic Web Approach to Service Description for Matchmaking of Services". In Proceedings of the 1st Semantic Web Working Symposioum. Callifornia 2001. pp. 447-461.

[32]. Gunnar Astrand Grimnes, Stuart Chalmers, Pete Edwards, Alun Preece (2003). "GraniteNights- A multi-Agent Visit Sceduler Utilizing Semantic Web Technology". In Proceedings of the 7th International Workshop on Cooperative Information Systems (CIA-03). August 2003 Helsinki Finland.

Agent Technology

[33]. Tim Finin, Richard Fritzson, Don McKay, Robin MacEntire (1994). "KQML as an Agent Communication Language".

[34]. Yannis Labrou, Tim Finin, Yun Peng (1999). "Agent Communication Languages: The current Landscape". IEEE Intelligent Systems Journal March/April 1999.

[35]. F.Bellifemine, G Caire, A.Poggi, G. Rimassa (2003). "JADE a White Paper» Telecom Italia EXP magazine Vol 3, No 3 September 2003.

[36]. Chris van Aart, Ruurd Pels, Giovanni Caire, Federico Bergenti (2002). "Creating and Using Ontologies in Agent Communication". Telecom Italia EXP magazine Vol 2, No 3 September 2002.

[37]. Brent K Langley, Massimo Paolucci, Katia Sycara (2003). "Discovery of infrastructure in Multi-Agent Systems". In Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS- 2003).

[38]. Roberto A. Flores-Mendez (1999). "Towards a Standardization of Multi-Agent System Frameworks". ACM Crossroads Journal – Intelligent Agents. Summer 1999.

[39]. Youyong Zou, Tim Finin, Li Ding, Harry Chen, Rong Pan (2003) "Using Semantic Web Technology in Multi-Agent Systems: A Case Study in the TAGA trading Agent Environment". In Proccedings of the 5th International Conference on Electronic Commerce. Pittsburgh, Pennsylvania 2003. pp. 95-101.

[40]. Katia Sycara (1998). "Multi-Agent Systems". Artificial Intelligence Magazine 19(2). 1998

[41]. H-C. Wong and K. Sycara (2000). "A Taxonomy of Middle-agents for the Internet". In Proceedings of the 4th International Conference on Multi Agent Systems (ICMAS-2000).

[42]. Project JXTA: an Open Innovative Collaboration. Sun Microsystems.

[43]. Jaime Delgado, Isabel Gallego, Roberto Garcia, Rosa Gil (2002). "An Architecture for Negotiation with Mobile Agents". In Proceedings of the 4th International Workshop on Mobile Agents for Telecommunication Applications. pp. 21-32.

Semantic Web

[44]. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis and K. Tolle (2001). "The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases". In Proceedings of the 2nd International Workshop on the Semantic Web, Hongkong, May 1, 2001.

[45]. Grigoris Antoniou, Frank van Harmelen (2004). "A Semantic Web Primer". MIT Press 2004.

[46]. D. Beckett (2004). RDF/XML Syntax Specification, W3C Recommendation, February 2004. Available at: http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[47]. T. Berners-Lee (1999). "Weaving the Web". Harper 1999.

[48]. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler (2000). Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation, October 2000. Available at: http://www.w3.org/TR/2000/REC-xml-20001006.

[49]. Dan Brickley, R.V. Guha (2004). RDF Vocabulary Description Language 1.0: RDF Schema W3C Recommendation, February 2004. Available at: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/ .

[50]. D.L. McGuinness , F. van Harmelen (2004). OWL Web Ontology Language Overview W3C Recommendation, February 2004. Available at: http://www.w3.org/TR/owl-features/.

[51]. P. Haase, J. Broekstra, A. Eberhart, R. Volz (2004). "A Comparison of RDF Query Languages". In Proccedings of the 3rd International Semantic Web Conference. Japan 2004.

[52]. Ian Dickinson (2004). "The Semantic Web and Software Agents: Partners or Just Neighbours"? Agentlink Newsletter. September 2004.

Logic - Defeasible Reasoning

[53]. G. Antoniou, D. Billington, G. Governatori and M.J. Maher (2001). "Representation Results for Defeasible Logic". ACM Transactions on Computational Logic 2, 2 (2001): 255 – 287.

[54]. G. Antoniou, A. Bikakis (2004). "A System for Non-Monotonic Rules on the Web». Workshop on Rules and Rule Markup Languages for the Semantic Web (RuleML 2004), G. Antoniou, H. Boley (Ed.), Springer-Verlag, Hiroshima, Japan, 8 Nov. 2004.

[55]. G. Antoniou, M. J. Maher and D. Billington (2000). "Defeasible Logic versus Logic Programming without Negation as Failure". Journal of Logic Programming 41,1.

[56]. Nick Bassiliades, Grigoris Antoniou, Ioannis Vlahavas (2004). "A Defeasible Logic Reasoner for the Semantic Web». Workshop on Rules and Rule Markup Languages for the Semantic Web (RuleML 2004), G. Antoniou, H. Boley (Ed.), Springer-Verlag, Hiroshima, Japan, 8 Nov. 2004.

[57]. G. Governatori and M.J. Maher (2000). "An Argumentation-theoretic Characterization of Defeasible Logic". In Proceedings of the 14th European Conference on Artificial Intelligence, Amsterdam, 2000. IOS Press.

[58]. Guido Governatori, Arthur H.M ter Hofstede, Philipa Oaks (2000). "Defeasible Logic for Automated Negotiation".

[59]. B. N. Grosof (1997). "Prioritized Conflict Handing for Logic Programs". In Proceedings of the 1997 International Symposium on Logic Programming, 197-211.

[60]. M.J. Maher (2001). "Propositional Defeasible Logic has Linear Complexity". Theory and Practice of Logic Programming 1,6, p. 691-711.

[61]. M.J. Maher (2002). "A Model-Theoretic Semantics for Defeasible Logic", In Proceedings of Workshop on Paraconsistent Computational Logic, 67 - 80, 2002.

[62]. M.J. Maher, A. Rock, G. Antoniou, D. Billington and T. Miller (2001). "Efficient Defeasible Reasoning Systems". International Journal of Tools with Artificial Intelligence 10,4 (2001): 483-501.

[63]. D. Nute (1994). "Defeasible Logic". In Handbook of logic in artificial intelligence and logic programming (vol. 3): nonmonotonic reasoning and uncertain reasoning. Oxford University Press.

[64]. A.H.M. ter Hofstede (1993). "Information Modeling in Data Intensive Domains". PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993.

[65]. J.J. van Griethuysen, editor (1982). "*Concepts* and Terminology for the Conceptual Schema and the Information Base". Publ. nr. ISO/TC97/SC5/WG3-N695, ANSI, 11 West 42nd Street, New York, NY 10036, 1982.

Mischelaneous

[66]. FIPA specification http://www.fipa.org/repository/index.html.

[67]. Agent Construction Tools http://www.agentbuilder.com/AgentTools/index.html.

[68]. Java Agent Development http://jade.cselt.it

[69]. Protege Ontology and Knowledge-Base editor http://protege.stanford.edu/ .

[70]. Nguyen T. Giang, Dang T. Tung (2002). "Agent Platform Evaluation and Comparison".Pellucid 5FP IST-2001-34519.

[71]. Stefan Saroiu, P. Krishna, Gummadi, Steven D. Gribble (2002). "A Measurement Study of Peer-to-Peer File Sharing Systems" SPIE and ACM Multimedia - Multimedia Computing and Networking (MMCN-2002).

[72]. FIPA Interaction Protocols Specification (2002). http://www.fipa.org/repository/ips.php3.

[73]. FIPA ACL Message Structure Specification (2002). http://www.fipa.org/specs/fipa00061/SC00061G.html.

[74]. FIPA Communicative Act Library Specification (2002). http://www.fipa.org/specs/fipa00037/SC00037J.html.

[75]. FIPA Request Interaction Protocol Specification (2002). http://www.fipa.org/specs/fipa00026/SC00026H.html.

[76]. FIPA SL Content Language Specification (2002). http://www.fipa.org/specs/fipa00008/SC00008I.html.

[77]. FIPA RDF Content Language Specification (2001). http://www.fipa.org/specs/fipa00011/XC00011B.html.

[78]. E-Bay http://www.ebay.com.

[79]. Net Perception http://www.netperception.com.

[80]. CDNOW http://www.cdnow.com.

[81]. VRP. The ICS-FORTH Validating RDF Parser – VRP (2004). http://139.91.183.30:9090/RDF/VRP/.

Appendix A

Travel Domain Ontology

```
<?xml version="1.0" ?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   >
<rdfs:Class rdf:ID="Itinerary">
  <rdfs:comment>1. The class of the travel packages offered by the travel agents</rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="TravelAgent">
  <rdfs:comment>2. The class of the travel agents. </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="Service">
  <rdfs:comment>3. The class of the services included into a travel package.</rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="LocalTransportation">
  <rdfs:comment>4. The class of the available mean of transport at the place of  holiday.  </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Service"/>
</rdfs:Class>
<rdfs:Class rdf:ID="CarRentalCompany">
  <rdfs:comment>5. The class of Car rental companies. </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#LocalTransportation"/>
</rdfs:Class>
<rdfs:Class rdf:ID="MeansOfTransport">
  <rdfs:comment>6. The class of the available means of transport.  </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="Airline">
  <rdfs:comment>7. The class of Airline Companies.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#MeansOfTransport"/>
</rdfs:Class><rdfs:Class rdf:ID="Ferry">
  <rdfs:comment>8. The class of Ferry Companies.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#MeansOfTransport"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Railway">
  <rdfs:comment>9. The class of Railway Companies. </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#MeansOfTransport"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Resort">
  <rdfs:comment>10. The class of available places to live during holiday.</rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="Hotel">
  <rdfs:comment>11. The class of Hotels.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Resort"/>
</rdfs:Class>
<rdfs:Class rdf:ID="RentRooms">
  <rdfs:comment>12. The class of Rooms for Rent.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Resort"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Place">
  <rdfs:comment>13. The class of places for holiday.</rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="Island">
  <rdfs:comment>14. The class of Islands.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Place"/>
</rdfs:Class>
<rdfs:Class rdf:ID="ContinentPlace">
  <rdfs:comment>15. The class of continent places.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Place"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Vehicle">
```

```
  <rdfs:comment>16. The class of Vehicles. </rdfs:comment>
</rdfs:Class>
<rdf:Property rdf:ID="hasVehicle">
   <rdfs:comment>   It relates particular vehicle to different carRentalCompanies</rdfs:comment>
   <rdfs:domain rdf:resource="#CarRentalCompany"/>
   <rdfs:range rdf:resource="#Vehicle"/>
</rdf:Property>

<rdf:Property rdf:ID="offeredBy">
  <rdfs:comment> It relates particular itineraries to different travel agents </rdfs:comment>
  <rdfs:domain rdf:resource="#Itinerary"/>
  <rdfs:range rdf:resource="#TravelAgent"/>
</rdf:Property>
<rdf:Property rdf:ID="includesService">
  <rdfs:comment>   It relates a service to a particular itinerary</rdfs:comment>
  <rdfs:domain rdf:resource="#Itinerary"/>
  <rdfs:range rdf:resource="#Service"/>
</rdf:Property>
<rdf:Property rdf:ID="includesTransportation">
  <rdfs:comment> It relates particular means of transport to particular itinerary</rdfs:comment>
  <rdfs:domain rdf:resource="#Itinerary"/>
  <rdfs:range rdf:resource="#MeansOfTransport"/>
</rdf:Property>
<rdf:Property rdf:ID="includesResort">
  <rdfs:comment> It relates a resort to a particular itinerary</rdfs:comment>
  <rdfs:domain rdf:resource="#Itinerary"/>
  <rdfs:range rdf:resource="#Resort"/>
</rdf:Property>
<rdf:Property rdf:ID="forPlace">
  <rdfs:comment>It relates particular place of holiday to particular itinerary</rdfs:comment>
  <rdfs:domain rdf:resource="#Itinerary"/>
  <rdfs:range rdf:resource="#Place"/>
</rdf:Property>
<rdf:Property rdf:ID="resortID">
  <rdfs:comment> A unique identifier for a resort</rdfs:comment>
  <rdfs:domain rdf:resource="#Resort"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="garden">
  <rdfs:comment> A boolean indicator for the existence of garden</rdfs:comment>
  <rdfs:domain rdf:resource="#Resort"/>
  <rdfs:range rdf:resource="&xsd;boolean"/>
</rdf:Property>
<rdf:Property rdf:ID="swimmingPool">
  <rdfs:comment> A boolean indicator for the existence of pool</rdfs:comment>
  <rdfs:domain rdf:resource="#Resort"/>
  <rdfs:range rdf:resource="&xsd;boolean"/>
</rdf:Property>
<rdf:Property rdf:ID="parking">
  <rdfs:comment>A boolean indicator for the existence of parking</rdfs:comment>
  <rdfs:domain rdf:resource="#Resort"/>
  <rdfs:range rdf:resource="&xsd;boolean"/>
</rdf:Property>
<rdf:Property rdf:ID="breakfast">
  <rdfs:comment>A boolean indicator for the existence of breakfast</rdfs:comment>
  <rdfs:domain rdf:resource="#Resort"/>
  <rdfs:range rdf:resource="&xsd;boolean"/>
</rdf:Property>
<rdf:Property rdf:ID="distanceFromSea">
  <rdfs:comment> The distance of the resort from the sea</rdfs:comment>
  <rdfs:domain rdf:resource="#Resort"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
<rdf:Property rdf:ID="hotelName">
  <rdfs:comment> Hotel name</rdfs:comment>
  <rdfs:domain rdf:resource="#Hotel"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="hotelStars">
  <rdfs:comment> The category of the hotel expressed in stars</rdfs:comment>
```

```
  <rdfs:domain rdf:resource="#Hotel"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
<rdf:Property rdf:ID="hotelCategory">
  <rdfs:comment> The category of the hotel expressed verbally e.g. Business Class etc. </rdfs:comment>
  <rdfs:domain rdf:resource="#Hotel"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="country">
  <rdfs:comment>The place of vocation</rdfs:comment>
  <rdfs:domain rdf:resource="#Place"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="islandName">
  <rdfs:comment> The name of the island</rdfs:comment>
  <rdfs:domain rdf:resource="#Island"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="islandPopulation">
  <rdfs:comment>  The population of the island</rdfs:comment>
  <rdfs:domain rdf:resource="#Island"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
<rdf:Property rdf:ID="price">
  <rdfs:comment> The price of the itinerary</rdfs:comment>
  <rdfs:domain rdf:resource="#Itinerary"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
<rdf:Property rdf:ID="persons">
  <rdfs:comment> The number of persons for the itinerary</rdfs:comment>
  <rdfs:domain rdf:resource="#Itinerary"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
<rdf:Property rdf:ID="from">
  <rdfs:comment> Place of departure</rdfs:comment>
  <rdfs:domain rdf:resource="#Itinerary"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="to">
  <rdfs:comment> Place of holiday</rdfs:comment>
  <rdfs:domain rdf:resource="#Itinerary"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="departureDate">
  <rdfs:comment> Date of departure</rdfs:comment>
  <rdfs:domain rdf:resource="#Itinerary"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="returnDate">
  <rdfs:comment> Place of departure</rdfs:comment>
  <rdfs:domain rdf:resource="#Itinerary"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="taID">
 <rdfs:comment> A unique identifier for the travel agent</rdfs:comment>
 <rdfs:domain rdf:resource="#TravelAgent"/>
 <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="taName">
  <rdfs:comment> The name of the travel agent company</rdfs:comment>
  <rdfs:domain rdf:resource="#TravelAgent"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="crcName">
  <rdfs:comment> The name of the car rental company</rdfs:comment>
  <rdfs:domain rdf:resource="#CarRentalCompany"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="serviceID">
  <rdfs:comment> A unique identifier of an offered service</rdfs:comment>
```

```
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="airlineName">
  <rdfs:comment> Airline Name</rdfs:comment>
  <rdfs:domain rdf:resource="#Airline"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="ferryName">
  <rdfs:comment>Ferry Name</rdfs:comment>
  <rdfs:domain rdf:resource="#Ferry"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="railwayName">
  <rdfs:comment>Airline Name</rdfs:comment>
  <rdfs:domain rdf:resource="#Railway"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="vehicleName">
  <rdfs:comment> Vehicle Name</rdfs:comment>
  <rdfs:domain rdf:resource="#Vehicle"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="vehicleCategory">
  <rdfs:comment>Vehicle Category </rdfs:comment>
  <rdfs:domain rdf:resource="#Vehicle"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="vehicleCC">
  <rdfs:comment>Vehicle cc </rdfs:comment>
  <rdfs:domain rdf:resource="#Vehicle"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
<rdf:Property rdf:ID="vehicleAC">
  <rdfs:comment>A binary indicator for the existence of A/C </rdfs:comment>
  <rdfs:domain rdf:resource="#Vehicle"/>
  <rdfs:range rdf:resource="&xsd;boolean"/>
</rdf:Property>
<rdf:Property rdf:ID="rentPricePerDay">
  <rdfs:comment> Price fof vehicle for rental </rdfs:comment>
  <rdfs:domain rdf:resource="#Vehicle"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
</rdf:RDF>
```

Travel Ontology Instances

```
<?xml version="1.0"?>
 <!DOCTYPE rdf:RDF [
 <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
 <!ENTITY schema "http://www.csd.uoc.gr/~dogjohn/ontology/TravelOntology.rdfs#">]
>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:schema ="http://www.csd.uoc.gr/~dogjohn/ontology/TravelOntology.rdfs#">

<schema:Hotel rdf:ID="CretaMareRoyal">
</schema:Hotel>
<rdf:Description rdf:about="#CretaMareRoyal">
  <rdf:type rdf:resource="&schema;Hotel"/>
  <schema:resortID rdf:datatype="&xsd;string">1</schema:resortID>
  <schema:hotelName rdf:datatype="&xsd;string">Creta Mare Royal</schema:hotelName>
  <schema:hotelStars rdf:datatype="&xsd;integer">6</schema:hotelStars>
  <schema:hotelCategory rdf:datatype="&xsd;string">Business</schema:hotelCategory>
  <schema:parking rdf:datatype="&xsd;boolean">true</schema:parking>
  <schema:swimmingPool rdf:datatype="&xsd;boolean">true</schema:swimmingPool>
  <schema:breakfast rdf:datatype="&xsd;boolean">true</schema:breakfast>
```

```
  <schema:distanceFromSea rdf:datatype="&xsd;integer">10</schema:distanceFromSea>
</rdf:Description>
<schema:Hotel rdf:ID="KnwssosPalace">
</schema:Hotel>
<rdf:Description rdf:about="#KnwssosPalace">
 <rdf:type rdf:resource="&schema;Hotel"/>
  <schema:resortID rdf:datatype="&xsd;string">2</schema:resortID>
  <schema:hotelName rdf:datatype="&xsd;string">Knwssos Palace</schema:hotelName>
  <schema:hotelStars rdf:datatype="&xsd;integer">4</schema:hotelStars>
  <schema:hotelCategory rdf:datatype="&xsd;string">Business</schema:hotelCategory>
  <schema:parking rdf:datatype="&xsd;boolean">true</schema:parking>
  <schema:swimmingPool rdf:datatype="&xsd;boolean">false</schema:swimmingPool>
  <schema:breakfast rdf:datatype="&xsd;boolean">false</schema:breakfast>
  <schema:distanceFromSea rdf:datatype="&xsd;integer">1000</schema:distanceFromSea>
</rdf:Description>
<schema:Hotel rdf:ID="SkopelosBay">
</schema:Hotel>
<rdf:Description rdf:about="#SkopelosBay">
  <rdf:type rdf:resource="&schema;Hotel"/>
  <schema:resortID rdf:datatype="&xsd;string">3</schema:resortID>
  <schema:hotelName rdf:datatype="&xsd;string">Skopelos Bay</schema:hotelName>
  <schema:hotelStars rdf:datatype="&xsd;integer">4</schema:hotelStars>
  <schema:hotelCategory rdf:datatype="&xsd;string">Business</schema:hotelCategory>
  <schema:parking rdf:datatype="&xsd;boolean">false</schema:parking>
  <schema:swimmingPool rdf:datatype="&xsd;boolean">false</schema:swimmingPool>
  <schema:breakfast rdf:datatype="&xsd;boolean">true</schema:breakfast>
  <schema:distanceFromSea rdf:datatype="&xsd;integer">1200</schema:distanceFromSea>
</rdf:Description>
<schema:Hotel rdf:ID="MykonosGrecotel">
</schema:Hotel>
<rdf:Description rdf:about="#MykonosGrecotel">
  <rdf:type rdf:resource="&schema;Hotel"/>
  <schema:resortID rdf:datatype="&xsd;string">4</schema:resortID>
  <schema:hotelName rdf:datatype="&xsd;string">Mykonos Imperial Grecotel</schema:hotelName>
  <schema:hotelStars rdf:datatype="&xsd;integer">6</schema:hotelStars>
  <schema:hotelCategory rdf:datatype="&xsd;string">Luxury</schema:hotelCategory>
  <schema:parking rdf:datatype="&xsd;boolean">true</schema:parking>
  <schema:swimmingPool rdf:datatype="&xsd;boolean">true</schema:swimmingPool>
  <schema:breakfast rdf:datatype="&xsd;boolean">true</schema:breakfast>
  <schema:distanceFromSea rdf:datatype="&xsd;integer">5</schema:distanceFromSea>
</rdf:Description>
<schema:Island rdf:ID="Creta">
</schema:Island>
<rdf:Description rdf:about="#Creta">
  <rdf:type rdf:resource="&schema;Island"/>
  <schema:islandName rdf:datatype="&xsd;string">Creta</schema:islandName>
  <schema:islandPopulation rdf:datatype="&xsd;integer">250000</schema:islandPopulation>
</rdf:Description>
<schema:Island rdf:ID="Skopelos">
</schema:Island>
<rdf:Description rdf:about="#Skopelos">
  <rdf:type rdf:resource="&schema;Island"/>
  <schema:islandName rdf:datatype="&xsd;string">Skopelos</schema:islandName>
  <schema:islandPopulation rdf:datatype="&xsd;integer">4000</schema:islandPopulation>
</rdf:Description>
<schema:Island rdf:ID="Mykonos">
</schema:Island>
<rdf:Description rdf:about="#Mykonos">
 <rdf:type rdf:resource="&schema;Island"/>
  <schema:islandName rdf:datatype="&xsd;string">Mykonos</schema:islandName>
  <schema:islandPopulation rdf:datatype="&xsd;integer">7000</schema:islandPopulation>
</rdf:Description>
<schema:Island rdf:ID="Kefalonia">
</schema:Island>
<rdf:Description rdf:about="#Kefalonia">
  <rdf:type rdf:resource="&schema;Island"/>
  <schema:islandName rdf:datatype="&xsd;string">Kefalonia</schema:islandName>
  <schema:islandPopulation rdf:datatype="&xsd;integer">16000</schema:islandPopulation>
</rdf:Description>
<schema:Airline rdf:ID="Olympic">
```

```
</schema:Airline>
<rdf:Description rdf:about="#Olympic">
  <rdf:type rdf:resource="&schema;Airline"/>
  <schema:airlineName
rdf:datatype="&xsd;string">Olympic Airways S.A.</schema:airlineName>
</rdf:Description>
<schema:Airline rdf:ID="Aegean">
</schema:Airline>
<rdf:Description rdf:about="#Aegean">
  <rdf:type rdf:resource="&schema;Airline"/>
  <schema:airlineName
rdf:datatype="&xsd;string">Aegean Airways S.A.</schema:airlineName>
</rdf:Description>
<schema:Ferry rdf:ID="Minoan">
</schema:Ferry>
<rdf:Description rdf:about="#Minoan">
  <rdf:type rdf:resource="&schema;Ferry"/>
  <schema:ferryName rdf:datatype="&xsd;string">Minoan Lines</schema:ferryName>
</rdf:Description>

<schema:Ferry rdf:ID="Anek">
</schema:Ferry>
<rdf:Description rdf:about="#Anek">
  <rdf:type rdf:resource="&schema;Ferry"/>
  <schema:ferryName rdf:datatype="&xsd;string">Anek Lines</schema:ferryName>
</rdf:Description>
<schema:Ferry rdf:ID="BS">
</schema:Ferry>
<rdf:Description rdf:about="#BS">
  <rdf:type rdf:resource="&schema;Ferry"/>
  <schema:ferryName
rdf:datatype="&xsd;string">Blue Star Ferries </schema:ferryName>
</rdf:Description>
<schema:Ferry rdf:ID="STR">
</schema:Ferry>
<rdf:Description rdf:about="#STR">
  <rdf:type rdf:resource="&schema;Ferry"/>
  <schema:ferryName
rdf:datatype="&xsd;string">Strintzis Lines</schema:ferryName>
</rdf:Description>
<schema:TravelAgent rdf:ID="ZORP">
</schema:TravelAgent>
<rdf:Description rdf:about="#ZORP">
 <rdf:type rdf:resource="&schema;TravelAgent"/>
 <schema:taName rdf:datatype="&xsd;string">Zorpidis</schema:taName>
</rdf:Description>
<schema:TravelAgent rdf:ID="INTERTOUR">
</schema:TravelAgent>
<rdf:Description rdf:about="#INTERTOUR">
 <rdf:type rdf:resource="&schema;TravelAgent"/>
 <schema:taName
rdf:datatype="&xsd;string">Interschema Travel Services S.A</schema:taName>
</rdf:Description>
<schema:TravelAgent rdf:ID="POSEIDON">
</schema:TravelAgent>
<rdf:Description rdf:about="#POSEIDON">
 <rdf:type rdf:resource="&schema;TravelAgent"/>
 <schema:taName rdf:datatype="&xsd;string">Poseidon Travel S.A.</schema:taName>
</rdf:Description>
<schema:Vehicle rdf:ID="Cherokee">
</schema:Vehicle>
<rdf:Description rdf:about="#Cherokee">
  <rdf:type rdf:resource="&schema;Vehicle"/>
  <schema:vehicleName
rdf:datatype="&xsd;string">Grand Cherokee</schema:vehicleName>
  <schema:vehicleCategory
rdf:datatype="&xsd;string">Jeep</schema:vehicleCategory>
  <schema:vehicleCC rdf:datatype="&xsd;integer">4000</schema:vehicleCC>
  <schema:vehicleAC rdf:datatype="&xsd;boolean">true</schema:vehicleAC> <schema:rentPricePerDay
rdf:datatype="&xsd;integer">120</schema:rentPricePerDay>
```

```
</rdf:Description>
<schema:Vehicle rdf:ID="Saxo">
</schema:Vehicle>
<rdf:Description rdf:about="#Saxo">
  <rdf:type rdf:resource="&schema;Vehicle"/>
  <schema:vehicleName
rdf:datatype="&xsd;string">Citroen Saxo vts</schema:vehicleName>
  <schema:vehicleCategory
rdf:datatype="&xsd;string">Car</schema:vehicleCategory>
  <schema:vehicleCC rdf:datatype="&xsd;integer">1300</schema:vehicleCC>
  <schema:vehicleAC rdf:datatype="&xsd;boolean">false</schema:vehicleAC>
  <schema:rentPricePerDay rdf:datatype="&xsd;integer">60</schema:rentPricePerDay>
</rdf:Description>
<schema:Vehicle rdf:ID="Astra">
</schema:Vehicle>
<rdf:Description rdf:about="#Astra">
  <rdf:type rdf:resource="&schema;Vehicle"/>
  <schema:vehicleName rdf:datatype="&xsd;string">Opel Astra</schema:vehicleName>
</rdf:Description>
<schema:CarRentalCompany rdf:ID="AVIS">
</schema:CarRentalCompany >
<rdf:Description rdf:about="#AVIS">
  <rdf:type rdf:resource="&schema;CarRentalCompany"/>
  <schema:crcName
rdf:datatype="&xsd;string">AVIS car rental ltd.</schema:crcName>
  <schema:hasVehicle rdf:resource="#Cherokee"/>
</rdf:Description>
<schema:CarRentalCompany rdf:ID="CretaCar">
</schema:CarRentalCompany>
<rdf:Description rdf:about="#CretaCar">
  <rdf:type rdf:resource="&schema;CarRentalCompany"/>
  <schema:crcName rdf:datatype="&xsd;string">Creta car rental S.A.</schema:crcName>
  <schema:hasVehicle rdf:resource="#Astra"/>
</rdf:Description>
</rdf:RDF>
```

Travel Package Advertisements

Advertisement 1

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
            <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
            <!ENTITY instances "http://www.csd.uoc.gr/~dogjohn/data/TravelOntologyInstances.rdf#">
            <!ENTITY schema "http://www.csd.uoc.gr/~dogjohn/ontology/TravelOntology.rdfs#">
           ]
>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:instances ="&instances;"
   xmlns:schema="&schema;"
>
<schema:Itinerary rdf:ID="IT1">
</schema:Itinerary>
<rdf:Description rdf:about="#IT1">
 <rdf:type rdf:resource="&schema;Itinerary"/>
  <schema:from rdf:datatype="&xsd;string">thessaloniki</schema:from>
  <schema:to rdf:datatype="&xsd;string">Creta</schema:to>
  <schema:departureDate rdf:datatype="&xsd;string">3/8/2004</schema:departureDate>
  <schema:returnDate rdf:datatype="&xsd;string">15/8/2004</schema:returnDate>
  <schema:persons rdf:datatype="&xsd;integer">2</schema:persons>
  <schema:price rdf:datatype="&xsd;integer">1000</schema:price>
```

```
  <schema:offeredBy rdf:resource="&instances;ZORP"/>
  <schema:includesResort rdf:resource="&instances;CretaMareRoyal"/>
  <schema:includesTransportation rdf:resource="&instances;Minoan"/>
  <schema:includesService rdf:resource="&instances;AVIS"/>
  <schema:forPlace rdf:resource="&instances;Creta"/>
</rdf:Description>
</rdf:RDF>
```

Advertisement 2

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
        <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
        <!ENTITY instances "http://www.csd.uoc.gr/~dogjohn/data/TravelOntologyInstances.rdf#">
        <!ENTITY schema "http://www.csd.uoc.gr/~dogjohn/ontology/TravelOntology.rdfs#">

]>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:instances ="&instances;"
   xmlns:schema="&schema;"
>
<schema:Itinerary rdf:ID="IT2">
</schema:Itinerary>
<rdf:Description rdf:about="#IT2">
 <rdf:type rdf:resource="&schema;Itinerary"/>
 <schema:from rdf:datatype="&xsd;string">athens</schema:from>
  <schema:to rdf:datatype="&xsd;string">Skopelos</schema:to>
  <schema:departureDate rdf:datatype="&xsd;string">3/8/2004</schema:departureDate>
  <schema:returnDate rdf:datatype="&xsd;string">15/8/2004</schema:returnDate>
  <schema:persons rdf:datatype="&xsd;integer">4</schema:persons>
  <schema:price rdf:datatype="&xsd;integer">1800</schema:price>
  <schema:offeredBy rdf:resource="&instances;TERTOUR"/>
  <schema:includesResort rdf:resource="&instances;SkopelosBay"/>
  <schema:includesTransportation rdf:resource="&instances;Minoan"/>
  <schema:includesService rdf:resource="&instances;AVIS"/>
  <schema:forPlace rdf:resource="&instances;Skopelos"/>
</rdf:Description>
</rdf:RDF>
```

Advertisement 3

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
        <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
        <!ENTITY instances "http://www.csd.uoc.gr/~dogjohn/data/TravelOntologyInstances.rdf#">
        <!ENTITY schema "http://www.csd.uoc.gr/~dogjohn/ontology/TravelOntology.rdfs#">

]>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:instances ="&instances;"
   xmlns:schema="&schema;"
>
<schema:Itinerary rdf:ID="IT3">
</schema:Itinerary>
<rdf:Description rdf:about="#IT3">
 <rdf:type rdf:resource="&schema;Itinerary"/>
 <schema:from rdf:datatype="&xsd;string">athens</schema:from>
  <schema:to rdf:datatype="&xsd;string">Mykonos</schema:to>
  <schema:departureDate rdf:datatype="&xsd;string">3/8/2004</schema:departureDate>
  <schema:returnDate rdf:datatype="&xsd;string">15/8/2004</schema:returnDate>
  <schema:persons rdf:datatype="&xsd;integer">2</schema:persons>
  <schema:price rdf:datatype="&xsd;integer">2500</schema:price>
  <schema:offeredBy rdf:resource="&instances;POSEIDON"/>
  <schema:includesResort rdf:resource="&instances;MykonosGrecotel"/>
```

```
  <schema:includesTransportation rdf:resource="&instances;Anek"/>
  <schema:includesService rdf:resource="&instances;AVIS"/>
  <schema:forPlace rdf:resource="&instances;Mykonos"/>
</rdf:Description>
</rdf:RDF>
```

Advertisement 4

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
            <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
            <!ENTITY instances "http://www.csd.uoc.gr/~dogjohn/data/TravelOntologyInstances.rdf#">
            <!ENTITY schema "http://www.csd.uoc.gr/~dogjohn/ontology/TravelOntology.rdfs#">

]>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:instances ="&instances;"
   xmlns:schema="&schema;" >
<schema:Itinerary rdf:ID="IT4">
</schema:Itinerary>
<rdf:Description rdf:about="#IT4">
  <rdf:type rdf:resource="&schema;Itinerary"/>
  <schema:from rdf:datatype="&xsd;string">athens</schema:from>
  <schema:to rdf:datatype="&xsd;string">Mykonos</schema:to>
  <schema:departureDate rdf:datatype="&xsd;string">1/8/2004</schema:departureDate>
  <schema:returnDate rdf:datatype="&xsd;string">20/8/2004</schema:returnDate>
  <schema:persons rdf:datatype="&xsd;integer">2</schema:persons>
  <schema:price rdf:datatype="&xsd;integer">3000</schema:price>
  <schema:offeredBy rdf:resource="&instances;POSEIDON"/>
  <schema:includesResort rdf:resource="&instances;MykonosGrecotel"/>
  <schema:includesTransportation rdf:resource="&instances;Minoan"/>
  <schema:includesService rdf:resource="&instances;AVIS"/>
  <schema:forPlace rdf:resource="&instances;Mykonos"/>
</rdf:Description>
</rdf:RDF>
```