University of Crete

Computer Science Department

A Semantic Peer-to-Peer Marketplace Hosting Trusted Negotiations Among Intelligent Agents

Ioannis Androulakis

Master's Thesis

Heraklion, October 2009

University of Crete
School of Science and Technology
Computer Science Department

A Semantic Peer-to-Peer Marketplace Hosting Trusted Negotiations Among Intelligent Agents

Ioannis Androulakis

A thesis submitted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

Author:

_____

Ioannis Androulakis, Computer Science Department

Supervisory Committee:

_____

Dimitris Plexousakis, Professor, Supervisor

_____

Christos Nikolaou, Professor, Member

_____

Yannis Tzitzikas, Assistant Professor, Member

Approved by:

_____

Panos Trahanias, Professor

Chairman of the Graduate Studies Committee

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. Over the last 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. The Semantic Web vision promises to handle these new issues. Electronic Brokering is a good candidate for taking up Semantic Web technology. We study the matchmaking problem that is, how a requester's requirements and preferences can be matched against a set of offerings collected by a broker and also check how trustworthy the providers of the offerings are. The proposed solution uses the Semantic Web standard of OWL to represent the offerings, and reactive rules for expressing the requirements and preferences. We also implement a negotiating agent architecture in the multi-agent environment JADE following the FIPA specifications that uses FIPA ACL messages for the communication between agents. The strategy of each agent is affected by various parameters and can change in many different ways from the beginning to the end of the negotiation. Agents also have the ability to detect abnormal bidding behavior during the negotiation process.

This thesis also describes a platform that defines and implements all the basic stages of the overall process of e-trading, facilitating human users in closing deals in an automated manner. The platform is based on a design that integrates three prominent technologies for addressing issues of next generation e-Commerce applications: autonomous intelligent software agents, peer-to-peer networking and the Semantic Web.

The implementation of our approach is demonstrated in the context of auction scenarios. The platform is designed to be highly extensible to allow experimentation with the technologies involved. A thorough evaluation of the system's performance, under various test cases, is demonstrated showing all the agent's abilities described above.

Supervisor: Plexousakis Dimitris
Professor

Μια Αγορά Αξιόπιστων Διαπραγματεύσεων μεταξύ Ευφυών Πρακτόρων Λογισμικού, Δομημένων σε ένα Διομότιμο Σημασιολογικό Δίκτυο

Ιωάννης Ανδρουλάκης

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 και επίσης γίνεται έλεγχος για το πόσο έμπιστος είναι ο πάροχος. Η προτεινόμενη λύση χρησιμοποιεί το μοντέλο δεδομένων OWL για την έκφραση των προσφορών και δυναμικούς κανόνες για την έκφραση των προτιμήσεων. Επίσης υλοποιούμε μια αρχιτεκτονική διαπραγμάτευσης πρακτόρων στο περιβάλλον πρακτόρων JADE η οποία ακολουθεί το αντίστοιχο πρότυπο της FIPA χρησιμοποιεί FIPA ACL μηνύματα για την επικοινωνία μεταξύ των πρακτόρων. Η στρατηγική του κάθε πράκτορα επιρεάζεται απο ποικιλία παραμέτρων και μπορεί να αλλάξει με διάφορους τρόπους από την αρχή μέχρι το τέλος της διαπραγμάτευσης. Ο κάθε πρακτορας έχει την δυνατότητα να εντοπίζει πάραλογες συμπεριφορές στις προσφορές των πελατών.

Η παρούσα εργασία περιγράφει μια πλατφόρμα η οποία ορίζει και υλοποιεί όλα τα βασικά στάδια της διαδικασίας της ηλεκτρονικης ανταλλαγης. Η πλατφόρμα βασίζεται σε μία σχεδίαση που ενοποιεί τρεις σημαντικές τεχνολογίες, για την διευθέτηση θεμάτων των εφαρμογών ηλεκτρονικού εμπορίου

επόμενης γενιάς: την τεχνολογία αυτόνομων, ευφυών πρακτόρων λογισμικού, τα ομότιμα δίκτυα και το Σημασιολογικό Ιστό.

Η υλοποίηση της προσέγγισής μας περιγράφεται στο πλαίσιο σεναρίων δημοπρασιών. Η πλατφόρμα σχεδιάστηκε να είναι ιδιαίτερα επεκτάσιμη, ώστε να επιτρέπει τον πειραματισμό με τις τεχνολογίες που εμπεριέχει. Για το λόγο αυτό και παρουσιάζεται μια εκτενής αξιολόγηση της απόδοσης του συστήματος υπό ποικίλες καταστάσεις ελέγχου δείχνοντας έτσι όλες τις ικανότητες των πρακτόρων που περιγράφηκαν παραπάνω.

Επόπτης: Πλεξουσάκης Δημήτριος
Καθηγητής

Ευχαριστίες

Καταρχήν θέλω να εκφράσω την ευγνωμοσύνη μου στον επόπτη μου, καθηγητή Παν/μίου Κρήτης κ. Δημήτρη Πλεξουσάκη που πρωτίστως με εμπιστεύτηκε και στη συνέχεια με καθοδήγησε δίνοντάς μου πολύτιμες συμβουλές ώστε να ολοκληρώσω με επιτυχία τις μεταπτυχιακές μου σπουδές.

Οφείλω να αναφερθώ στον υποψήφιο διδάκτορα Θοδωρή Πάτκο, που μου προσέφερε ανεκτίμητη βοήθεια και υλικό για θεωρητικά και πρακτικά ζητήματα, και με ανέχτηκε, παραμελώντας συχνά τις ακαδημαϊκές του υποχρεώσεις. Η βοήθεια του απο την στιγμή που ξεκίνησε η συνεργασία μας ενώ ακόμα ήμουν προπτυχιακός φοιτητής αποτέλεσε καθοριστίκο παράγοντα για να συνέχισω την προσπάθεια μου και να πίστεψω ότι μπόρω να τα καταφέρω.

Θα ήθελα επίσης να ευχαριστήσω την πολύ καλή φίλη που απέκτησα κατα την διάρκεια των μεταπτυχιακών μου σπουδών, απόφοιτη μεταπτυχιακή φοιτήτρια Μαρία Μίχου η οποία ήταν πάντα εκεί όχι μόνο για να με στηρίξει ψυχολογικά αλλά και πρακτικά έχοντας πάντα την διάθεση να με βοηθήσει σε οποιαδήποτε απορία μου. Το ενδιαφέρον της τον τελευταίο ειδικά χρόνο ήταν ένα γερό στήριγμα για να συνεχίσω να προσπαθώ και να ολοκληρώσω την εργασία μου πάντα μειώνοντας τους φόβους και τις ανασφάλειές μου.

Επιπρόσθετα, ευχαριστώ τους Μαρία, Νίκο, Βούλα, Μανόλη, Γωγώ, Έρη και γενικά όλους του φίλους και σύναδερφους που μου προσέφεραν ψυχολογική υποστήριξη την κατάλληλη στιγμή και έκαναν πιο ευχάριστη αυτή τη διαδρομή.

Ολοκληρώνοντας, θέλω να ευχαριστήσω τους γονείς μου Γιώργο και Μαρία και την αδερφή μου Ελίνα για την συμπαράστασή τους, που με στήριξαν και εξακολουθούν να με στηρίζουν σε όλα μου τα βήματα. Σας ευχαριστώ.

Στην αγαπημένη μου γιαγιά Ελένη

που δεν προλάβαμε να χαρούμε

για την ολοκλήρωση αυτής

της προσπάθειας μαζί ...

List Of Figures

List Of Tables

1 Introduction

1.1 Motivations and Goals

In the last years, there has been a great interest in electronic commerce (e-commerce) potential [11]. 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 [11]. 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.

Our work is based on the previous work of Patkos Theodoros who as part of his work on his master created the SeMPHoNIA platform [1] for addressing many of current e-trading issues. The SeMPHoNIA project attempts to proceed one step beyond the state-of-the-art. Our goal is to integrate and exploit all three technologies, namely intelligent software agents, peer-to-peer systems and the Semantic Web, into a unified platform for demonstrating the many benefits that arise from their collaboration. SeMPHoNIA is an architecture for an agent-based virtual marketplace and is intended as a platform for research in multi-agent systems, ontologies, peer-to-peer networking and automatic negotiation, as well as, an application for experimentation with these technologies. The platform defines and implements the basic stages of the overall process of e-trading by closing deals in a semi-automated manner, utilizing knowledge from queryable product repositories in an open peer-to-peer environment. SeMPHoNIA could be

considered as what [9] describes as the third key actor in agent-mediated e-Commerce applications, apart from buyers and sellers; the “market owner”, an environment that sets and controls the rules, in which buyers and sellers trade.

The first deference with SeMPHoNIA is that we use JADE[33] to create our platform. JADE has an advantage we exploit, the user must not create the peers but they are automatically created so this gives as the opportunity to join the agent and peer to peer part in one single part. Moreover another of the authors’ suggestions was the creation of graphics representing the auctions progress and this is another extra characteristic we gave to the platform. We also followed the authors’ suggestion and use the standardized communication language FIPA ACL [75] to allow heterogeneous agents to exchange information and knowledge.

A problem we also wanted a client to be able to solve was shilling. Shills, or "potted plants", are sometimes employed in auctions. Driving prices up with phony bids, they seek to provoke a bidding war among other participants. Often they are told by the seller precisely how high to bid, as the seller actually pays the price (to himself, of course) if the item does not sell, losing only the auction fees.

Shilling has a substantially higher rate of occurrence in online auctions, where any user with multiple accounts (and IP addresses) can shill without aid of participants. Many online auction sites employ sophisticated (and usually secret) methods to detect collusion.

We propose a simple yet expressive framework for giving the ability to negotiating agents to find shill behaviours and manage to not be tricked by shillers. Specifically, we explore the suitability of reactive rules for expressing the decision-making process of negotiating agents in such cases. To validate the viability of our approach, we apply it to negotiation scenarios and provide a prototype implementation of such agent architecture using JADE multiagent framework.

As an alternative to these techniques we propose the use of a technique for brokering. In particular, advertisements are expressed in a web ontology language (OWL) and user preferences and requests are captured by the use of a dynamic graphic that is created with the help of OWL-API and transformed into a reactive rule that searches and gives the results. The brokering process is also affected by information the client gathers from other trusted agents and give him feedback about the previous actions of Auction Agents that affect the trust of the client agent and the money he is willing to give for buying an item in a specific auction. We provide a prototype implementation of a brokering system architecture again using the JADE multiagent framework.

A client has the ability to select to begin a number of negotiations at the same time either wanting to win all of them re win only one of them. In the former case the agents participate in all the negotiations trying to win. In the latter case

an agent is created and follows a strategy that allows him to win in only one negotiation, the aggressiveness of the agent is affected by various parameters like the trust of the other agents, the maximum amount of money he is willing to give to an agent, the remaining time of a negotiation and the number of negotiations he is taking part in.

The implementation of our approach is demonstrated in the context of auction scenarios following the example of SeMPHoNIA. On-line auctions represent a more specific type of negotiation governed by predefined protocols and have rapidly achieved enormous popularity as a common interaction medium both for humans and software agents on the internet. Users are offered the ability to participate in multiple auctions at the same time, when the result of one auction may affect the action taken for the other and the agent follows a dynamic strategy that takes under consideration the trust level, the remaining time and the number of auctions in order to achieve the best offer. The platform is intended to facilitate users in discovering and bidding across multiple auctions with varying start and end times and varying protocols, attempting to ensure that at most one of the desired items will be purchased agreeing the best, for the customer, deal. Our implementation currently supports two types of auctions, namely English and Vickrey . The platform utilizes peer-to-peer architecture and semantic components as a mechanism for auction discovery, as well as, intelligent agents for trust management, participation and automatic negotiation in these auctions. Still, auction scenarios are just a paradigm of multi-agent negotiation. Our intention is to encourage researchers and application designers to explore and experiment with aspects of other domains that go beyond auction theory, such as automated negotiation in general, Semantic Web, e-Commerce, publication and discovery of resources and Web Services etc.

1.2 Description of the remaining chapters

The rest structure of this thesis is organized as follows: In Chapter 2 the background theory is discussed that is needed to understand the main aspects of this thesis. In Chapter 3 related work is reviewed in the field of trust management and negotiations. In Chapter 4, the system architecture is described and the tools we used in order to develop it. In Chapter 5 some interesting characteristics of our system are analyzed. Chapter 6 shows how the auctions services are created and Chapter 7 gives details about the implementation of the services and the characteristics described. In Chapter 8 application scenarios are presented in order to highlight the functionalities of the system and in Chapter 9 some experiments are analyzed showing the abilities of the negotiation algorithm we used and why our approach is better than others. Finally, in chapter 10, we

present a final conclusion about our work and future improvements and extensions of the system are discussed.

2 Background

This section reviews the basic terms and principles of technologies used in this thesis. It explains the general features of software agents, Semantic Web and peer-to-peer systems and describes existing Internet-based auction sites, identifying their limitations. In addition, a survey of related work is presented.

2.1 Agent technology

2.1.1 What is it?

Agent-based systems are one of the most vibrant and important areas of research and development to have emerged in information technology in the 1990s. Put at its simplest, an agent is a computer system that is capable of flexible autonomous action in dynamic, unpredictable, typically multi-agent domains. In particular, the characteristics of dynamic and open environments in which, for example, heterogeneous systems must interact, span organizational boundaries, and operate effectively within rapidly changing circumstances and with dramatically increasing quantities of available information, suggest that improvements on traditional computing models and paradigms are required. Thus, the need for some degree of autonomy, to enable components to respond dynamically to changing circumstances while trying to achieve over-arching objectives, is seen by many as fundamental. Many observers therefore believe that agents represent the most important new paradigm for software development since object orientation.

The concept of an agent has found currency in a diverse range of sub-disciplines of information technology, including computer networks, software engineering, artificial intelligence, human-computer interaction, distributed and concurrent systems, mobile systems, telematics, computer supported cooperative work, control systems, decision support, information retrieval and management, and electronic commerce. In practical developments, web services, for example, now offer fundamentally new ways of doing business through a set of standardized tools, and support a service-oriented view of distinct and independent software components interacting to provide valuable functionality. In the context of such developments, agent technologies have increasingly come to the foreground. Because of its horizontal nature, it is likely

that the successful adoption of agent technology will have a profound, long-term impact both on the competitiveness and viability of IT industries, and on the way in which future computer systems will be conceptualized and implemented. Agent technologies can be considered from three perspectives, each outlined below, as illustrated in Figure 1.

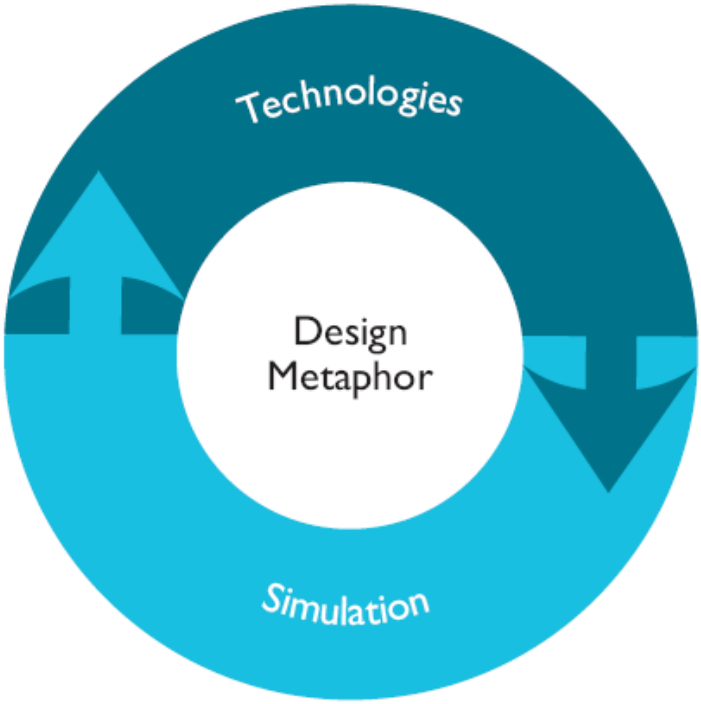


Figure 1 The three perspectives of Agent technologies

2.1.1.1 Agents as Design Metaphor

Agents provide software designers and developers with a way of structuring an application around autonomous, communicative components, and lead to the construction of software tools and infrastructure to support the design metaphor. In this sense, they offer a new and often more appropriate route to the development of complex computational systems, especially in open and dynamic environments. In order to support this view of systems development, particular tools and techniques need to be introduced. For example, methodologies to guide analysis and design are required, agent architectures are needed for the design of individual software components, tools and abstractions are required to enable developers to deal with the complexity of implemented systems, and supporting infrastructure (embracing other relevant, widely used technologies, such as web services) must be integrated.

2.1.1.2 Agents as a Source of Technologies

Agent technologies span a range of specific techniques and algorithms for dealing with interactions in dynamic, open environments. These address issues such as balancing reaction and deliberation in individual agent architectures,

learning from and about other agents in the environment, eliciting and acting upon user preferences, finding ways to negotiate and cooperate with other agents, and developing appropriate means of forming and managing coalitions (and other organizations). Moreover, the adoption of agent-based approaches is increasingly influential in other domains. For example, multi-agent systems are already providing new and more effective methods of resource allocation in complex environments than previous approaches.

2.1.1.3 Agents as Simulation Tool

Multi-agent systems offer strong models for representing complex and dynamic real-world environments. For example, simulation of economies, societies and biological environments are typical application areas.

The use of agent systems to simulate real-world domains may provide answers to complex physical or social problems that would otherwise be unobtainable due to the complexity involved, as in the modeling of the impact of climate change on biological populations, or modeling the impact of public policy options on social or economic behaviour. Agent based simulation spans: social structures and institutions to develop plausible explanations of observed phenomena, to help in the design of organizational structures, and to inform policy or managerial decisions; physical systems, including intelligent buildings, traffic systems and biological populations; and software systems of all types, currently including eCommerce and information management systems.

In addition, multi-agent models can be used to simulate the behaviour of complex computer systems, including multi-agent computer systems. Such simulation models can assist designers and developers of complex computational systems and provide guidance to software engineers responsible for the operational control of these systems. Multi-agent simulation models thus effectively provide a new set of tools for the management of complex adaptive systems, such as large-scale online resource allocation environments.

We do not claim that agent systems are simply panaceas for these large problems; rather they have been demonstrated to provide concrete competitive advantages such as:

→Improving operational robustness with intelligent failure recovery;

→ Reducing sourcing costs by computing the most beneficial acquisition policies in online markets; and

→ Improving efficiency of manufacturing processes in dynamic environments.

2.1.2 Agent technologies Levels, Tools and techniques

It should be clear that there are several distinct high-level trends and drivers leading to interest in agent technologies, and low-level computing infrastructures making them practically feasible. In this context, we now consider the key technologies and techniques required to design and implement agent systems that are the focus of current research and development. Because agent technologies are mission-critical for engineering and for managing certain types of information systems, such as Grid systems ([15]. [16].) and systems for

ambient intelligence ([14]), the technologies and techniques discussed below will be important for many applications, even those not labeled as agent systems.

These technologies can now be grouped into three categories, according to the scale at which they apply:

→ Organization-level: At the top level are technologies and techniques related to agent societies as a whole. Here, issues of organizational structure, trust, norms and obligations, and self-organization in open agent societies are paramount. Once again, many of these questions have been studied in other disciplines — for example, in sociology, anthropology and biology. Drawing on this related work, research and development are currently focused on technologies for designing, evolving and managing complex agent societies.

→ Interaction-level: These are technologies and techniques that concern the communications between agents — for example, technologies related to communication languages, interaction protocols and resource allocation mechanisms. Many of the problems solved by these technologies have been studied in other disciplines, including economics, political science, philosophy and linguistics. Accordingly, research and development is drawing on this prior work to develop computational theories and technologies for agent interaction, communication and decision-making.

→ Agent-level: These are technologies and techniques concerned only with individual agents — for example, procedures for agent reasoning and learning. Problems at this level have been the primary focus of artificial intelligence since its inception, aiming to build machines that can reason and operate autonomously in the world. Agent research and development has drawn extensively on this prior work, and most attention in the field of agent-based computing now focuses at the previous two higher levels.

In addition to technologies at these three levels, we must also consider technologies providing infrastructure and supporting tools for agent systems, such as agent programming languages and software engineering methodologies. These supporting technologies and techniques provide the basis for both the theoretical understanding and the practical implementation of agent systems.

2.1.2.1 Organization Level

Organizations

Dynamic agent organizations that adjust themselves to gain advantage in their current environments are becoming increasingly important over the last five years. They arise in dynamic (or emergent) agent societies, such as those suggested by the Grid, ambient intelligence and other domains in which agents come together to deliver composite services, all of which require that agents can adapt to function effectively in uncertain or hostile environments. Some work has already started on the development of systems that can meet this challenge, which is fundamental to realizing the power of the agent paradigm; its relevance will remain at the forefront of R&D efforts over the next 10- 15 years, especially in relation to commercial efforts at exploitation. In particular, building dynamic agent

organizations (including, for example, methods for teamwork, coalition formation, and so on) for dealing with aspects of the emerging visions of the Grid and the Web, as well as aspects of ubiquitous computing, will be crucial.

Social factors in the organization of multi-agent systems will also become increasingly important over the next decade as we seek ways to structure interactions in an open and dynamic online world. This relates to the need to properly assign roles, (institutional) powers, rights and obligations to agents in order to control security and trust-related aspects of multi-agent systems at a semantic level, as opposed to current developments, which deal with them at the infrastructure level. These social factors provide the basis on which to develop methods for access control, for example, and to ensure that behaviour is regulated and structured when faced with dynamic environments in which traditional techniques are not viable. In addition to appropriate methods and technologies for agent team formation, management, assessment, coordination and dissolution, technologies will also be required for these processes to be undertaken automatically at runtime in dynamic environments.

Complex Systems and Self Organization

Self-organization refers to the process by which a system changes its internal organization to adapt to changes in its goals and environment without explicit external control. This can often result in emergent behaviour that may or may not be desirable. Due to the dynamism and openness of contemporary computing environments, understanding the mechanisms that can be used to model, assess and engineer self organization and emergence in multi-agent systems is an issue of major interest.

A self-organizing system functions through contextual local interactions, without central control. Components aim to individually achieve simple tasks, but a complex collective behaviour emerges from their mutual interactions. Such a system modifies its structure and functionality to adapt to changes to requirements and to the environment based on previous experience. Nature provides examples of self-organization, such as ants foraging for food, molecule formation, and antibody detection. Similarly, current software applications involve social interactions (such as negotiations and transactions) with autonomous entities or agents, in highly dynamic environments. Engineering applications to achieve robustness and adaptability, based on the principles of self-organization, is thus gaining increasing interest in the software community. This interest originates from the fact that current software applications need to cope with requirements and constraints stemming from the increased dynamism, sophisticated resource control, autonomy and decentralization inherent in contemporary business and social environments. The majority of these characteristics and constraints are the same as those that can be observed in natural systems exhibiting self-organization.

Self-organization mechanisms provide the decision-making engines based on which system components process input from software and hardware sensors to decide how, when and where to modify the system's structure and functionality. This enables a better fit with the current requirements and environment, while preventing damage or loss of service. It is therefore

necessary to characterize the applications in which existing mechanisms, such as stigmergy (or the means by which the individual parts of a system communicate with one another by modifying their local environment, much like ants), can be used, and to develop new generic mechanisms independent of any particular application domain.

In some cases, self-organization mechanisms have been modeled using rule-based approaches or control theory. Furthermore, on many occasions the self-organizing actions have been inspired by biological and natural processes, such as the human nervous system and the behaviour observed in insect species that form colonies. Although such approaches to self-organization have been effective in certain domains, environmental dynamics and software complexity have limited their general applicability. More extensive research in modeling self-organization mechanisms and systematically constructing new ones is therefore needed. Future self-organizing systems must accommodate high dimensional sensory data, continue to learn from new experiences and take advantage of new self-organization acts and mechanisms as they become available.

A phenomenon is characterized as emergent if it has not been exactly predefined in advance. Such a phenomenon can be observed at a macro system level and it is generally characterized by novelty, coherence, irreducibility of macro level properties to micro-level ones and non-linearity. In multi-agent systems, emergent phenomena are the global system behaviours that are collective results originating from the local agent interactions and individual agent behaviours. Emergent behaviours can be desirable or undesirable; building systems with desirable emergent behaviour capabilities can increase their robustness, autonomy, openness and dynamism.

To achieve desired global emergent system behaviour, local agent behaviours and interactions should comply with some behavioural framework dictated by a suitable theory of emergence. Unfortunately, too few theories of emergence are currently available and existing ones still require improvement. In consequence, therefore, new theories of emergence need to be developed based on inspiration from natural or social systems, for example.

An important open issue in self-organizing systems relates to modeling the application context and environment. In this respect, a key question is the definition of the relevant environmental parameters that need to be considered in determining the evolving structure and functionality of self-organizing software. Additional open questions relate to: how context can be captured, processed and exploited for adjusting the services provided by the application in a given situation; how the self-organizing effects occurring from participation of the application in different contexts can be synchronized; how to effectively model user preferences and intentions; and the amount of historical information that should be recorded by the system and considered in determining its evolution over time.

Trust and Reputation

Many applications involving multiple individuals or organizations must take into account the relationships (explicit or implicit) between participants. Furthermore, individual agents may also need to be aware of these relationships in order to make appropriate decisions.

The field of trust, reputation and social structure seeks to capture human notions such as trust, reputation, dependence, obligations, permissions, norms, institutions and other social structures in electronic form.

By modeling these notions, engineers can borrow strategies commonly used by humans to resolve conflicts that arise when creating distributed applications, such as regulating the actions of large populations of agents using financial disincentives for breaking social rules or devising market mechanisms that are proof against certain types of malicious manipulation. The theories are often based on insights from different domains including economics (market-based approaches), other social sciences (social laws, social power) or mathematics (game theory and mechanism design).

The complementary aspect of this social perspective relating to reputation and norms is a traditional concern with security. Although currently deployed agent applications often provide good security, when considering agents autonomously acting on behalf of their owner several additional factors need to be addressed. In particular, collaboration of any kind, especially in situations in which computers act on behalf of users or organizations, will only succeed if there is trust. Ensuring this trust requires, for example, the use of: reputation mechanisms to assess prior behaviour; norms (or social rules) and the enforcement of sanctions; and electronic contracts to represent agreements.

Whereas assurance deals primarily with system integrity, security addresses protection from malicious entities: preventing would-be attackers from exploiting self-organization mechanisms that alter system structure and behaviour. In addition, to verify component sources, a self-organizing software system must protect its core from attacks. Various well-studied security mechanisms are available, such as strong encryption to ensure confidentiality and authenticity of messages related to self-organization. However, the frameworks within which such mechanisms can be effectively applied in self-organizing systems still require considerable further research.

In addition, the results of applying self-organization and emergence approaches over long time periods lead to concerns about the privacy and trustworthiness of such systems and the data they hold. The areas of security, privacy and trust are critical components for the next stages of research and deployment of open distributed systems and as a result of self-organizing systems. New approaches are required to take into account both social and technical aspects of this issue to drive the proliferation of self-organizing software in a large range of application domains.

2.1.2.2 Interaction Level

Coordination

Coordination is defined in many ways but in its simplest form it refers to ensuring that the actions of independent actors (agents) in an environment are coherent in some way ([13]). The challenge therefore is to identify mechanisms that allow agents to coordinate their actions automatically without the need for human supervision, a requirement found in a wide variety of real applications. In turn, cooperation refers to coordination with a common goal in mind.

Research to date has identified a huge range of different types of coordination and cooperation mechanisms, ranging from emergent cooperation (which can arise without any explicit communication between agents), coordination protocols (which structure interactions to reach decisions) and coordination media (or distributed data stores that enable asynchronous communication of goals, objectives or other useful data), to distributed planning (which takes into account possible and likely actions of agents in the domain).

Negotiation

Goal-driven agents in a multi-agent society typically have conflicting goals, in other words, not all agents may be able to satisfy their respective goals simultaneously. This may occur, for example, with regard to contested resources or with multiple demands on an agent's time and attention. In such circumstances, agents will need to enter into negotiations with each other to resolve conflicts. Accordingly, considerable effort has been devoted to negotiation protocols, resource-allocation methods, and optimal division procedures.

This work has drawn on ideas from computer science and artificial intelligence on the one hand, and the socio-economic sciences on the other.

For example, a typical objective in multi-agent resource allocation is to find an allocation that is optimal with respect to a suitable metric that depends, in one way or another, on the preferences of the individual agents in the system. Many concepts studied in social choice theory can be utilized to assess the quality of resource allocations. Of particular importance are concepts such as envy-freeness and equitability that can be used to model fairness considerations ([12]. [19].). These concepts are relevant to a wide range of applications. A good example is the work on the fair and efficient exploitation of Earth Observation Satellite resources carried out at ONERA, the French National Aeronautics Research Centre ([20].).

While much work on resource allocation has concentrated on centralized approaches, in particular combinatorial auctions ([21].), many applications are more naturally modeled as truly distributed or P2P systems where allocations emerge as a consequence of a sequence of local negotiation steps ([22].).

The centralized approach has the advantage of requiring only comparatively simple communication protocols. Furthermore, advances in the design of powerful algorithms for combinatorial auctions have had a strong impact on the research community ([23].). A challenge in the field of multi-agent resource allocation is to transfer these techniques to distributed resource allocation frameworks, which are not only important in cases where it may be difficult to find an agent that could take on the role of the auctioneer (for instance,

in view of its computational capabilities or its trustworthiness), but which also provide a test-bed for a wide range of agent-based techniques. To reach its full potential, distributed resource allocation requires further fundamental research into agent interaction protocols, negotiation strategies, formal (e.g. complexity-theoretic) properties of resource allocation frameworks, and distributed algorithm design, as well as a new perspective on what "optimal" means in a distributed setting.

Other negotiation techniques are also likely to become increasingly prevalent. For example, one-to-one negotiation, or bargaining, over multiple parameters or attributes to establish service-level agreements between service providers and service consumers will be key in future service-oriented computing environments. In addition to approaches drawn from economics and social choice theory in political science, recent efforts in argumentation based negotiation have drawn on ideas from the philosophy of argument and the psychology of persuasion. These efforts potentially provide a means to enable niches of deeper interactions between agents than do the relatively simpler protocols of economic auction and negotiation mechanisms. Considerable research and development efforts will be needed to create computational mechanisms and strategies for such interactions, and this is likely to be an important focus of agent systems research in the next decade.

Communication

Agent communication is the study of how two or more software entities may communicate with each other. The research issues in the domain are long-standing and deep. One challenge is the difficulty of assigning meaning to utterances, since the precise meaning of a statement depends upon: the context in which it is uttered; its position in a sequence of previous utterances; the nature of the statement (for example, a proposition, a commitment to undertake some action, a request, etc); the objects referred to in the statement (such as a real world object, a mental state, a future world-state, etc); and the identity of the speaker and of the intended hearers. Another challenge, perhaps insurmountable, is semantic verification: how to verify that an agent means what it says when it makes an utterance. In an open agent system, one agent is not normally able to view the internal code of another agent in order to verify an utterance by the latter; even if this were possible, a sufficiently-clever agent could always simulate any desired mental state when inspected by another agent.

Key to this area is the need to map the relevant theories in the domain, and to develop a unifying framework for them. In particular, a formal theory of agent languages and protocols is necessary, so as to be able to study language and protocol properties comprehensively, and to rigorously compare one language or protocol with another. In addition, progress towards understanding the applicability of different agent communication languages, content languages and protocols in different application domains is necessary for wider adoption of research findings.

2.1.2.3 Agent Level

Reasoning is a critical faculty of agents, but the extent to which it is needed is determined by context. While reasoning in general is important, in open environments there are some specific concerns relating to heterogeneity of agents, trust and accountability, failure handling and recovery, and societal change. Work must be continued on the representation of computational concepts for the norms, legislation, authorities, enforcement, and so forth, which can underpin the development and deployment of dynamic electronic institutions or other open multi-agent system. Similarly, current work on coalition formation for virtual organizations is limited, with such organizations largely static. The automation of coalition formation may be more effective at finding better coalitions than humans can in complex settings, and is required, for example, for Grid applications.

One enabler for this is negotiation, yet while there have already been significant advances and real-world applications, research into negotiation mechanisms that are more complex than auctions and game-theoretic mechanisms is still in its infancy. Research into argumentation mechanisms, for example, and the strategies appropriate for participants under them, is also needed before argumentation techniques will achieve widespread deployment. In addition, many virtual organizations will be required to make decisions collectively, aggregating in some fashion the individual preferences or decisions of the participants. Research on the application to agent societies of social choice theory from political science and sociology is also relatively new, and considerably more work is needed here. Both these topics were considered in the discussion on negotiation above.

Even though learning technology is clearly important for open and scalable multi-agent systems, it is still in early development. While there has been progress in many areas, such as evolutionary approaches and reinforcement learning, these have still not made the transition to real-world applications. Reasons for this can be found in the fundamental difficulty of learning, but also in problems of scalability and in user trust in self-adapting software. In the longer term, learning techniques are likely becoming a central part of agent systems, while the shorter term offers application opportunities in areas such as interactive entertainment, which are not safety-critical.

2.1.3 Infrastructure and Supporting Technologies

Any infrastructure deployed to support the execution of agent applications, such as those found in ambient and ubiquitous computing must, by definition, be long-lived and robust.

In the context of self-organizing systems, this is further complicated and new approaches supporting the evolution of the infrastructures, and facilitating their upgrade and update at runtime, will be required. Given the potentially vast collection of devices, sensors, and personalized applications for which agent systems and self-organization may be applicable, this update problem is significantly more complex than so far encountered.

More generally, middleware, or platforms for agent interoperability, as well as standards, is crucial for the medium-term development of agent systems.

2.1.3.1 Interoperability

At present, the majority of agent applications exist in academic and commercial laboratories, but is not widely available in the real world. The move out of the laboratory is now starting to happen, but a higher degree of automation than is currently available in dealing with knowledge management is needed for information agents. In particular, this demands new web standards that enable structural and semantic description of information; and services that make use of these semantic representations for information access at a higher level. The creation of common ontologies, thesauri or knowledge bases plays a central role here, and merits further work on the formal descriptions of information and, potentially, a reference architecture to support the higher level services mentioned above.

Distributed agent systems that adapt to their environment must both adapt individual agent components and coordinate adaptation across system layers (i.e. application, presentation and middleware) and platforms. In other words interoperability must be maintained across possibly heterogeneous agent components during and after self-organization actions and outcomes. Furthermore, agent components are likely to come from different vendors and hence the developer may need to integrate different self-organization mechanisms to meet an application's requirements. The problem is further complicated by the diversity of self-organization approaches applicable at different system layers. In many cases, even solutions within the same layer are often not compatible. Consequently, developers need tools and methods to integrate the operation of agent components across the layers of a single system, among multiple computing systems, as well as between different self-organization frameworks.

2.1.3.2 Agent Oriented Software Engineering

Despite a number of languages, frameworks, development environments, and platforms that have appeared in the literature, implementing multi-agent systems is still a complex task. In part, to manage multi-agent systems complexity, the research community has produced a number of methodologies that aim to structure agent development. However, even if practitioners follow such methodologies during the design phase, there are difficulties in the

implementation phase, partly due to the lack of maturity in both methodologies and programming tools. There are also difficulties in implementation due to: a lack of specialized debugging tools; skills needed to move from analysis and design to code; the problems associated with awareness of the specifics of different agent platforms; and in understanding the nature of what is a new and distinct approach to systems development.

In relation to open and dynamic systems, new methodologies for systematically considering self-organization are required. These methodologies should be able to provide support for all phases of the agent-based software engineering life-cycle, allowing the developer to start from requirements analysis, identify the aspects of the problem that should be addressed using self-organization and design and implement the self-organization mechanisms in the behaviour of the agent components. Such methodologies should also encompass techniques for monitoring and controlling the self-organizing application or system once deployed.

In general, integrated development environment (IDE) support for developing agent systems is rather weak, and existing agent tools do not offer the same level of usability as state-of-the-art object-oriented IDEs. One main reason for this is the previous unavoidable tight coupling of agent IDEs and agent platforms, which results from the variety of agent models, platforms and programming languages. This is now changing, however, with an increased trend towards modeling rather than programming.

With existing tools, multi-agent systems often generate a huge amount of information related to the internal state of agents, messages sent and actions taken, but there are not yet adequate methods for managing this information in the context of the development process. This impacts both dealing with the information generated in the system and obtaining this information without altering the design of the agents within it. Platforms like JADE provide general introspection facilities for the state of agents and for messages, but they enforce a concrete agent architecture that may not be appropriate for all applications. Thus, tools for inspecting any agent architecture, analogous to the remote debugging tools in current object-oriented IDEs, are needed, and some have started to appear the last 5 years. Extending this to address other issues related to debugging for organizational features, and for considering issues arising from emergence in self organizing systems will also be important in the longer term. The challenge is relevant now, but will grow in importance as the complexity of installed systems increases further. The inherent complexity of agent applications also demands a new generation of CASE tools to assist application designers in harnessing the large amount of information involved.

This requires providing reasoning at appropriate levels of abstraction, automating the design and implementation process as much as possible, and allowing for the calibration of deployed multi-agent systems by simulation and run-time verification and control. More generally, there is a need to integrate existing tools into IDEs rather than starting from scratch. At present there are many research tools, but little that integrates with generic development environments, such as Eclipse; such advances would boost agent development

and reduce implementation costs. Indeed, developing multi-agent systems currently involves higher costs than using conventional paradigms due to the lack of supporting methods and tools.

The next generation of computing system demands large numbers of interacting components, be they services, agents or otherwise. Current tools work well with limited numbers of agents, but are generally not yet suitable for the development of large-scale (and efficient) agent systems, nor do they offer development, management or monitoring facilities able to deal with large amounts of information or tune the behaviour of the system in such cases.

Metrics for agent-oriented software are also needed: engineering always implies some activity of measurement, and traditional software engineering already uses widely applied measuring methods to quantify aspects of software such as complexity, robustness and mean time between failures. However, the dynamic nature of agent systems, and the generally non-deterministic behaviour of self-organizing agent applications deem traditional techniques for measurement and evaluation inappropriate. Consequently, new measures and techniques for both quantitatively and qualitatively assessing and classifying multi-agent systems applications (be they self-organizing or not) are needed.

2.1.3.3 Agent Programming Languages

Most research in agent-oriented programming languages is based on declarative approaches, mostly logic based. Imperative languages are in essence inappropriate for expressing the high-level abstractions associated with agent systems design; however, agent-oriented programming languages should (and indeed tend to) allow for easy integration with (legacy) code written in imperative languages. From the technological perspective, the design and development of agent-based languages is also important. Currently, real agent-oriented languages (such as BDI-style ones) are limited, and used largely for research purposes; apart from some niche applications, they remain unused in practice. However, recent years have seen a significant increase in the maturity of such languages, and major improvements in the development platforms and tools that support them ([26].).

Current research emphasizes the role of multi-agent systems development environments to assist in the development of complex multi-agent systems, new programming principles to model and realize agent features, and formal semantics for agent programming languages to implement specific agent behaviours.

A programming language for multi-agent systems should respect the principle of separation of concerns and provide dedicated programming constructs for implementing individual agents, their organization, their coordination, and their environment. However, due to the lack of dedicated agent programming languages and development tools (as well as more fundamental concerns relating to the lack of clear semantics for agents, coordination, etc), the

construction of multi-agent systems is still a time-consuming and demanding activity.

One key challenge in agent-oriented programming is to define and implement some truly agent-oriented languages that integrate concepts from both declarative and object oriented programming, to allow the definition of agents in a declarative way, yet supported by serious monitoring and debugging facilities. These languages should be highly efficient, and provide interfaces to existing mainstream languages for easy integration with code and legacy packages. While existing agent languages already address some of these issues, further progress is expected in the short term, but thorough practical experimentation in real-world settings (particularly large-scale systems) will be required before such languages can be adopted by industry, in the medium to long term.

In addition to languages for single agents, we also need languages for high-level programming of multi-agent systems. In particular, the need for expressive, easy-to-use, and efficient languages for coordinating and orchestrating intelligent heterogeneous components is already pressing and, although much research is already being done, the development of an effective programming language for coordinating huge, open, scalable and dynamic multi-agent systems composed of heterogeneous components is a longer term goal.

2.1.3.4 Formal Methods

While the notion of an agent acting autonomously in the world is intuitively simple, formal analysis of systems containing multiple agents is inherently complex. In particular, to understand the properties of systems containing multiple actors, powerful modeling and reasoning techniques are needed to capture possible evolutions of the system. Such techniques are required if agents and agent systems are to be modeled and analyzed computationally.

Research in the area of formal models for agent systems attempts to represent and understand properties of the systems through the use of logical formalisms describing both the mental states of individual agents and the possible interactions in the system. The logics used are often logics of belief or other modalities, along with temporal modalities, and such logics require efficient theorem-proving or model-checking algorithms when applied to problems of significant scale. Recent efforts have used logical formalisms to represent social properties, such as coalitions of agents, preferences and game-type properties.

It is clear that formal techniques such as model checking are needed to test, debug and verify properties of implemented multi-agent systems. Despite progress, there is still a real need to address the issues that arise from differences in agent systems, in relation to the paradigm, the programming languages used, and especially the design of self-organizing and emergent behaviour. For the latter, a programming paradigm that supports automated checking of both functional and non-functional system properties may be needed. This would lead to the need to certify agent components for correctness with

respect to their specifications. Such a certification could be obtained either by selecting components that have already been verified and validated offline using traditional techniques such as inspection, testing and model checking or by generating code automatically from specifications. Furthermore, techniques are needed to ensure that the system still executes in an acceptable, or safe, manner during the adaptation process, for example using techniques such as dependency analysis or high level contracts and invariants to monitor system correctness before, during and after adaptation.

2.1.3.5 Simulation

As mentioned earlier, agent-based computing provides a means to simulate both natural and artificial systems, including agent-based computational systems themselves. Such simulation modeling is increasingly providing guidance to decision-makers in areas of medicine, social policy and industrial engineering, and assisting in the design, implementation and management of artificial and computational systems. However, for the full potential of agent-based (or individual-based) simulation models to be realized, a number of research and development challenges need to be met. First among these is the development of a rigorous theory of agent-based simulation. When should one stop refining a simulation model, for example? How many iterations of a randomized simulation model or scenarios are required in order to have confidence in the results? How much detail is required to be simulated in a model? How much trust should be placed in the results? How can we avoid over-interpretation of results with abstract or vague terms? The answers to these questions are likely to depend on the application domain, so a single, unified theory may be impossible to achieve. But efforts towards this goal are needed, not least because of the increasing reliance placed on simulation models in important public policy decisions.

Another major challenge relates to the development of agent-based simulation models involving cognitive and rational agents. In economic systems, for example, it has long been known that the expectations of individual actors may influence their behaviour, and thus the global properties of the system. How may these anticipatory and reflective aspects of real-world societies be modeled by agent based simulation models? The rapid growth of online resource allocation systems, such as Grid systems, makes this an important issue. If a computational Grid comprises intelligent computational users, many of whom base their decisions on their own economic models of the Grid operation itself, then the task of management is complicated immensely: statements and actions by the system manager may impact the beliefs and intentions of the participants, and thus impact system operations and performance. The challenge of managing user expectations in this way is well-known to governors of central banks, such as the European Central Bank, as they try to manage national monetary policy. The theory and practice of agent simulation models are not sufficiently mature to provide guidance to managers in this task.

2.1.3.6 User Interaction Design

In future complex system environments, human involvement is likely to become more important, yet this requires the exploration and understanding of several new possibilities, including: autonomy and improvisation (to deal with unforeseen events, such as those caused by the behaviour of human users); a standardized agent communication language with a powerful semantics to drive some of agent behaviour and facilitate integration of human users; social and organizational models for multi-agent systems, in which programs and humans can naturally interact (hybrid systems). In addition, as software becomes self organising to fit in a variety of contexts, a new set of issues concerning the interaction with users is created. A key question here is how people can interact with continuously changing software. Additional questions concern whether it would be valuable to try to design implicit interaction with applications operating on indirect sensor-based input and in that case how could users migrate from traditional explicit to future implicit interaction. In addition, questions of decision-making authority, responsibility, delegation and control arise with systems of agents acting on behalf of, or in collaboration with, human decision-makers in mixed initiative systems. If agents or multi-agent systems are themselves responsible for decisions, these issues become more problematic ([27]).

2.2 Peer-to-Peer Networks

In a P2P every participating node acts both as a client and as a server ("servent") and "pays" its participation by providing access to some of its resources, most frequently, processing power and/or disk space. Although this idea is common to all P2P systems they differ considerably in their underlying architecture. A coarse but intuitive definition of P2P is given by Clay Shirkey (The Accelerator Group) [24]:

Peer-to-peer is a class of applications that take advantage of resources storage, cycles, content, human presence available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, peer-to- peer nodes must operate outside the DNS and have significant or total autonomy of central servers.

Thus a P2P system can be viewed as an application-level Internet on top of the Internet. To quickly decide whether a given system is P2P in Shirkey's sense the following "litmus test" can be applied:

• Does the system give the nodes at the edges of the network significant autonomy?

• Does the system allow for variable connectivity and temporary network addresses?

More conceptually we can identify several principles underlying P2P systems:

• The principle of sharing resources: all P2P systems involve an aspect of resource sharing, where resources can be physical resources, such as disk space or network bandwidth, as well as, logical resources, such as services or different forms of knowledge. By sharing of resources applications can be realized which could not be set up by a single node. This was the driving motivation behind a P2P system such as Napster.

• The principle of decentralization: this is an immediate consequence of sharing of resources. Parts of the system or even the whole system are no longer operated centrally. Decentralization is in particular interesting in order to avoid single-point-of failures or performance bottlenecks in the system. Examples of fully decentralized systems are Gnutella and Freenet.

• The principle of self-organization: when a P2P system becomes fully decentralized then there exists no longer a node that can centrally coordinate its activities or a database to store global information about the system centrally. Therefore nodes have to self-organize themselves, based on whatever local information is available and interacting with locally reachable nodes (neighbors). The global behaviour then emerges as the result of all the local behaviours that occur.

In addition P2P system often take into account other qualitative requirements such as the interest of participants of remaining anonymous or the fact that nodes in a P2P system are usually unreliable.

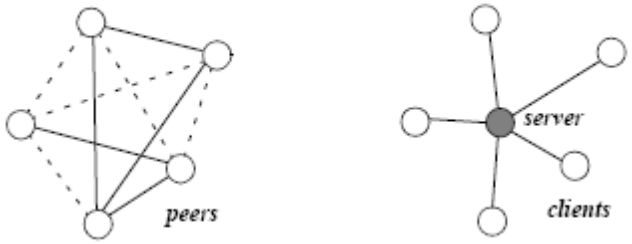


Figure 2 peer to peer and server/clients example

To understand better how and why an enhanced peer-to-peer architecture can play a better role in data communication and the Internet, an introduction to the traditional client/server architecture is presented.

Peer-to-peer (P2P) computing covers a wide range of infrastructures, technologies and applications that share a single characteristic: they are designed to create networked applications in which every node (or deployed system) is in some sense equivalent to all others, and application functionality is created by potentially arbitrary interconnection between these peers. The consequent absence of the need for centralized server components to manage P2P systems makes them highly attractive in terms of robustness against failure, ease of deployment, scalability and maintenance [18]. The best known P2P applications include hugely popular file sharing applications such as Gnutella and Bit Torrent, Akamai content caching, groupware applications (such as Groove Networks office environments) and Internet telephony applications such as Skype. While the majority of these well-known systems are based on proprietary protocols and platforms, toolkits such as Sun Microsystem's JXTA provide a wide array of networking features for the development of P2P applications, such as messaging, service advertisement and peer management features. Standardization for P2P technologies is also underway within the Global Grid Forum (GGF), which now includes a P2P working group established by Intel in 2000.

P2P applications display a range of agent-like characteristics, often applying selforganisation techniques in order to ensure continuous operation of the network, and relying on protocol design to encourage correct behaviour of clients. (For example, many commercial e-marketplace systems, such as eBay, include simple credit-reputation systems to reward socially beneficial behaviour). As P2P systems become more complex, an increasing number of agent technologies may also become relevant. These include, for example: auction mechanism design to provide a rigorous basis to incentivize rational behaviour

among clients in P2P networks; agent negotiation techniques to improve the level of automation of peers in popular applications; increasingly advanced approaches to trust and reputation; and the application of social norms, rules and structures, as well as social simulation, in order to better understand the dynamics of populations of independent agents.

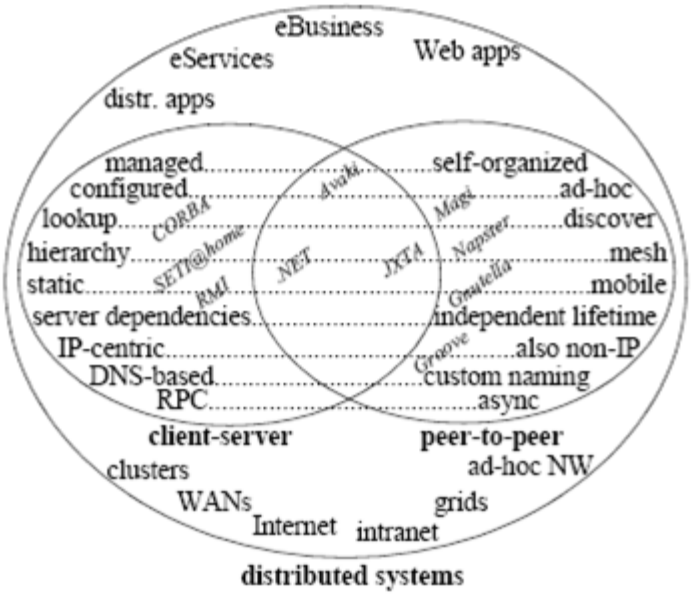
2.2.1 Traditional Client/Server Architecture

Most Internet services are distributed using the traditional client/server architecture. In this architecture, clients connect to a server using a specific communications protocol to obtain access to a specific resource. Most of the processing involved in delivering a service usually occurs on the server, leaving the client relatively unburdened. Most popular Internet applications, including the World Wide Web, FTP, telnet, and email, use this service-delivery model.

For a harmonic and synchronized communication and data transferring among members on a network there must be some protocol to be followed. Some of the most common network protocols are:

- TCP/IP: It is the basic protocol used for data communication on the Internet and is used for implementing many other protocols. In TCP/IP data are split and coded into small “packets”, which can be sent and received in a point-to-point architecture.
- HTTP: It is a convention in which Hypertext files (such as HTML files) can be downloaded from a server by clients. In Hypertext files each component (sentence or image) can point to other components and can also carry a runnable code as attachment (e.g. Java script files). These types of files should obey HTML for their structure. This method is very suitable for users with unreliable connections and also when the traffic is high, because clients change the server fast.
- FTP: This is a protocol designed for file transfer on the Internet. In this protocol any kind of file can be transferred between two nodes regardless of its content. An FTP server works faster than an HTTP server but it gives much less options to users.

Distributed systems, most of which currently work based on the client/server architecture, are an important means of data communication. Their main characteristic is that the components (hardware and software) are scattered on the network, offering resource sharing, parallel processing, higher reliability, extensibility etc. Some of the most popular technologies for developing distributed systems today involve socket, Remote Procedure Call (RPC), Java Remote Method Invocation (RMI), Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), Java Database Connectivity (JDBC), JINI.



Peer-to-Peer versus Client-Server. *There is no clear border between a client-server and a P2P model. Both models can be built on a spectrum of levels of characteristics (e.g., manageability, configurability), functionality (e.g., look-up versus discovery), organizations (e.g., hierarchy versus mesh), components (e.g., DNS), and protocols (e.g., IP), etc. Furthermore, one model can be built on top of the other or parts of the components can be realized in one or the other model. Finally, both models can execute on different types of platforms (Internet, intranet, etc.) and both can serve as an underlying base for traditional and new applications. Therefore, it should not be a surprise that there is so much confusion about what P2P is and what it is not. It is extremely intertwined with existing technologies [Morgan 2002].*

Figure 3 Peer to peer VS Client Server

2.2.2 Peer-to-Peer Architectures

Peer-to-peer (P2P) is one of the networked applications that users, called peers, can retrieve objects directly from each other without mediation of servers. Peers organize a logical network with logical links that correspond to the relationship among two peers that are in a neighborhood. In the view of client-server model, each peer is a server, a client, and a node at the same time. For example, when a peer permits other peers to download an MP3 file from its hard disk, the peer plays the role of a server. The peer also obtains files from other peers as a client. When messages exchanged among peers are relayed on logical P2P networks, a peer behaves as a node. For a peer to obtain an object, it must first find a peer that can provide it with the desired object. Thus, we need

mechanisms for a peer to discover other peers and determine which peers have the desired objects. Current P2P applications, most of which are P2P file sharing systems, have different mechanisms for dealing with such peer discovery and content location problems.

Here we employ the way in [25] to classify P2P applications into three basic architectures.

1. Centralized Directory

In this architecture, a central server provides a directory service for the all of peers. When a peer launches a P2P file-sharing application, the application first summarizes a list of objects in a local hard disk. Then it registers the list with a peer ID, IP address, and other type of information into a well-known central server, to which it establishes a permanent TCP connection. The server maintains a directory database that maps an object name to a set of IP addresses of provider peers. The directory database is kept up-to-date by receiving transaction messages. When a peer adds or removes an object, it informs the directory server of such changes. When a peer leaves the P2P network, the server detects the disappearance through the termination of the TCP connection.

To find an object, a peer sends a query message including some keywords and the maximum number of results it expects to the directory server. The server examines its directory database to find records of the object that contains the specified keywords. If there are, the server generates a list of records that match the query. On receiving the list, the peer selects a peer from which it directly retrieves objects.

Napster, OpenNap, and instant messaging applications such as ICQ, Yahoo messenger, and MSN messages are typical examples of this class.

2. Decentralized Directory

In this architecture, peers are clustered into groups. The entire logical P2P network consists of groups. A newly launched peer first inquires from a group whether it can join a bootstrapping node. Then, it becomes a member of the specified group. In a group, there is a leader that plays the role of a directory server of the group. It maintains information of the objects deposited by peers in the group. Thus, the mechanisms including registration, query, object retrieve in a group are similar to the “Centralized Directory” architecture. Additionally, to achieve more results, one leader peer can forward queries from its client peers to another leader peer. Of course, some mechanisms are needed to decide who becomes the leader of a group. A leader peer is chosen statically among peers in a group, or dynamically by an algorithm taking peer’s performance into account. Morpheus, KaZaA, eDonkey2000, and WinMX are typical applications of this class.

3. Query Flooding

In this architecture, all peers are equal. Since there is no directory server, this architecture is also called “pure P2P” while the others are called “hybrid P2P”. There is no hierarchical structure in a P2P network. To join a P2P network, a peer first sends a request to a bootstrapping node to provide it with a list of IP addresses of peers that have already participated in the network. Then the new peer advertises its address to those peers. Consequently, a neighborhood is built among them.

To find an object, a peer conducts “query flooding”. A peer begins flooding all its neighboring peers with query messages. To restrict the range in which the message propagates, a TTL (Time To Live) is appropriately set. Each of the neighboring peers first examines its local disk to determine whether it has the requested object. If it does, the peer sends back a response message to the requesting peer through an inverse way that a corresponding request message traversed. Before further relaying the query message, it decreases TTL. If TTL is larger than zero, the peer sends the message to all of its neighboring peers except for one from which the message originated. Gnutella and Freenet are typical applications using the query flooding mechanism.

Table 1 summarizes some advantages and disadvantages of the above three typical P2P architectures.

We can see the fact that all of today’s P2P architectures require some always-up nodes. Independent of architectures, there are still bottlenecks and single points of failure.

P2P architectures	Advantages	Disadvantages
Centralized directory	1. A peer can easily locate an object by sending a query to a directory server. 2. It is easy to manage and maintain the directory.	1. A directory server is a single point of failure. 2. A directory server is a performance bottleneck. 3. The architecture lacks scalability.
Decentralized directory	1. The load of directory service is distributed. 2. Failure of a directory server only affects a group of peers.	1. A complex algorithm and mechanism are necessary to construct a hierarchical network. 2. Distributed directory servers are single points of failure of their groups and bottlenecks. 3. A bootstrapping node is necessary and is a single point of failure.
Query flooding	1. All peers are equal. 2. No server is needed for searching.	1. A search involves much communication and traffic. 2. A bootstrapping node is necessary and is a single point of failure. 3. A complex protocol is needed to maintain a logical P2P network.

Table 1 advantages and disadvantages of P2P architectures.

2.3 Semantic Web

As discussed in [17], the Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation. The first steps in weaving the Semantic Web into the structure of the existing Web are already under way. In the near future, these developments will usher in significant new functionality as machines become much better able to process and "understand" the data that they merely display at present.

The essential property of the World Wide Web is its universality. The power of a hypertext link is that "anything can link to anything." Web technology, therefore, must not discriminate between the scribbled draft and the polished performance, between commercial and academic information, or among cultures, languages, media and so on. Information varies along many axes. One of these is the difference between information produced primarily for human consumption and that produced mainly for machines. At one end of the scale we have everything from the five-second TV commercial to poetry. At the other end we

have databases, programs and sensor output. To date, the Web has developed most rapidly as a medium of documents for people rather than for data and information that can be processed automatically. The Semantic Web aims to make up for this.

Like the Internet, the Semantic Web will be as decentralized as possible. Such Web-like systems generate a lot of excitement at every level, from major corporation to individual user, and provide benefits that are hard or impossible to predict in advance. Decentralization requires compromises: the Web had to throw away the ideal of total consistency of all of its interconnections, ushering in the infamous message "Error 404: Not Found" but allowing unchecked exponential growth.

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. 4, 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.

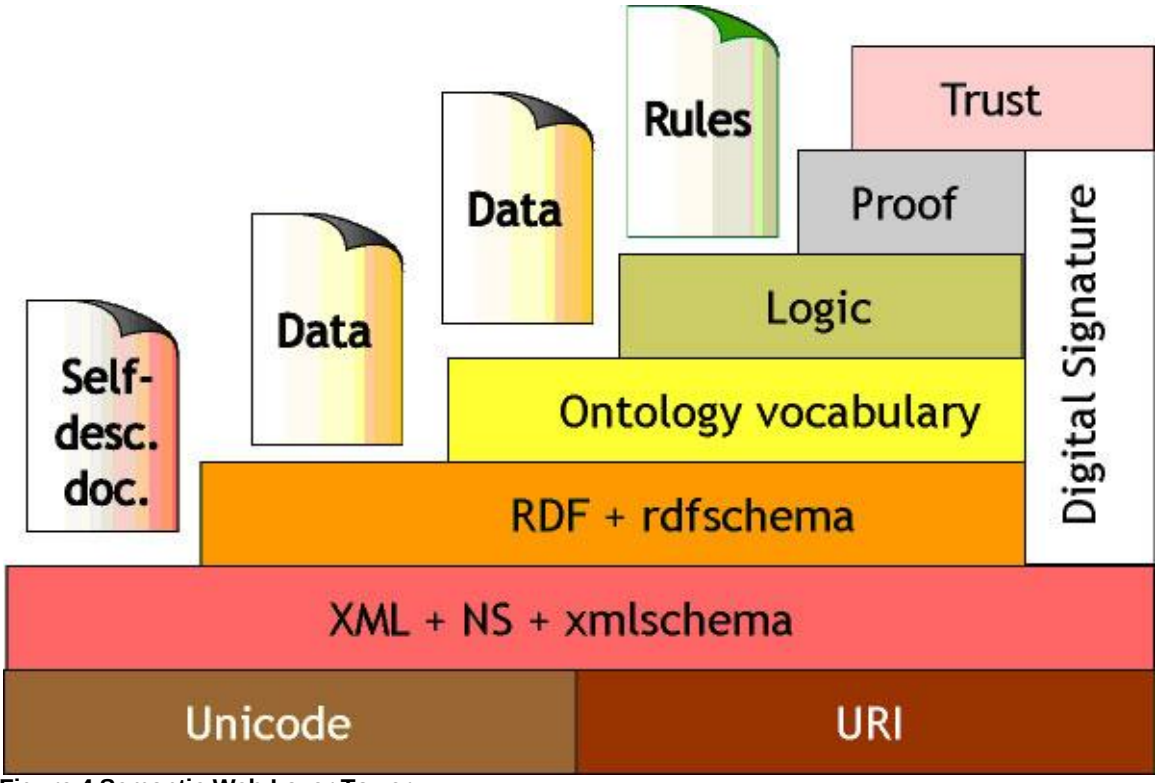


Figure 4 Semantic Web Layer Tower

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.

2.3.1 XML Basic Features

XML [28], 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.

2.3.2 RDF Basic Features

RDF [29] 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. 5. This kind of graph is known as a Semantic Net in the artificial intelligence community.

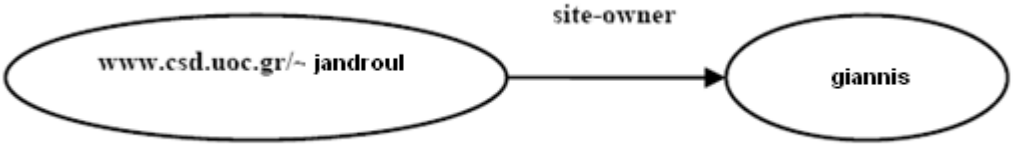


Figure 5 graph based representation

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. 6.

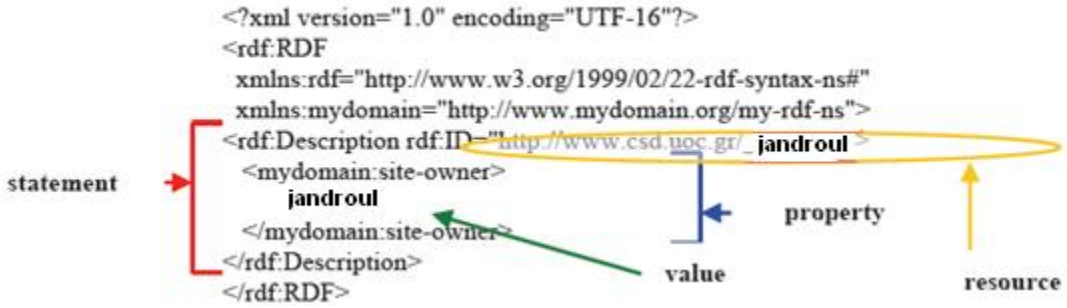


Figure 6 XML based representation

2.3.3 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) [30]. 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.

2.3.4 OWL Basic Features

OWL (Web Ontology Language) [31] comes to fill the missing features of RDF and RDFS. According to [32], 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”).

2.4 Rules and Rule Engines

In computer programming, too, it's important to choose the right tool for the right job. Some problems can be solved easily using traditional programming techniques, whereas writing a rule-based system is the easiest way to solve others. Other problems are in the middle, and either technique will work equally well.Much of the programming we do is procedural. Rule-based programming, however, is declarative.

In procedural programming, the programmer tells the computer what to do, how to do it, and in what order. Procedural programming is well suited for problems in which the inputs are well specified and for which a known set of steps can be carried out to solve the problem. Mathematical computations, for example, are best written procedurally. Note that here procedural is used in a slightly different way than is conventional.

Although object-oriented programming is traditionally contrasted with the older procedural programming, for the purposes of this discussion, the two are equivalent. In both procedural and object-oriented programming, the programmer writes all the logic that controls the computer, although it is partitioned differently in an object-oriented program. A purely declarative program, in contrast, describes what the computer should do, but omits much of the instructions on how to do it. Declarative programs must be executed by some kind of runtime system that understands how to fill in the blanks and use the declarative information to solve problems. Because declarative programs include only the important details of a solution, they can be easier to understand than procedural programs. Furthermore, because the control flow is chosen by the runtime system, a declarative program can be more flexible in the face of fragmentary or poorly conditioned. Declarative programming is often the natural way to tackle problems involving control, diagnosis, prediction, classification, pattern recognition, or situational awareness— in short, many problems without clear algorithmic solutions.

A rule-based program doesn't consist of one long sequence of instructions; instead, it is made up of discrete rules, each of which applies to some subset of the problem. In a rule-based program, you write only the individual rules. Another program, the rule engine, determines which rules apply at any given time and executes them as appropriate. As a result, a rule-based version of a complex program can be shorter and easier to understand than a procedural version. Writing the program is simpler, because you can concentrate on the rules for one situation at a time. Modifying the program is also simpler—if you've ever had to work on a program containing a dozen levels of nested if statements, you'll understand why. In the following paragraphs, we'll formalize some of the ideas behind rulebased systems and see how they are constructed.

2.4.1Expert systems

Expert systems, rule-based computer programs that capture the knowledge of human experts in their own fields of expertise, were a success story for artificial intelligence research in the 1970s and 1980s. Early, successful expert systems were built around rules (sometimes called heuristics) for medical diagnosis, engineering, chemistry, and computer sales. One of the early expert system successes was MYCIN,2 a program for diagnosing bacterial infections of the blood. Expert systems had a number of perceived advantages over human experts. For instance, unlike people, they could perform at peak efficiency, 24 hours a day, forever. There are numerous dramatic examples in the computer science literature of these early systems matching or exceeding the performance of their human counterparts in specific, limited situations. Predictions were made that someday, sophisticated expert systems would be able to reproduce general human intelligence and problem-solving abilities. Over time, of course, the drama receded, and it became clear that researchers had vastly underestimated the complexity of the common-sense knowledge that underpins general human reasoning. Nevertheless, excellent applications for expert systems remain to this day. Modern expert systems advise salespeople, scientists, medical technicians, engineers, and financiers, among others.

Today, general rule-based systems, both those intended to replace human expertise and those intended to automate or codify business practices or other activities, are a part of virtually every enterprise. These systems are routinely used to order supplies, monitor industrial processes, prescreen résumés, route telephone calls, and process web forms. Many commercial application servers incorporate a rule engine, and most others explicitly or implicitly offer integration with one. Expert systems really have become ubiquitous—we just don't call them by that name anymore.

2.4.2 Architecture of a rule-based system

The rules in the first expert systems were intertwined with the rest of the software, so that developing a new expert system meant starting from the ground up. The folks who wrote MYCIN, recognizing this fact, created a development tool named EMYCIN.3 EMYCIN (Empty MYCIN) was developed by removing all the medical knowledge from MYCIN, leaving behind only a generic framework for rule-based systems. EMYCIN was the first expert system shell. An expert system shell is just the inference engine and other functional parts of an expert system with all the domain-specific knowledge removed. Most modern rule engines can be seen as more or less specialized expert system shells, with features to support operation in specific environments or programming in specific domains. This book is about this kind of rule engine.

A typical rule engine contains:

■ An inference engine

■ A rule base

■ A working memory

The inference engine, in turn, consists of:

■ A pattern matcher

■ An agenda

■ An execution engine

These components are shown schematically in figure 7

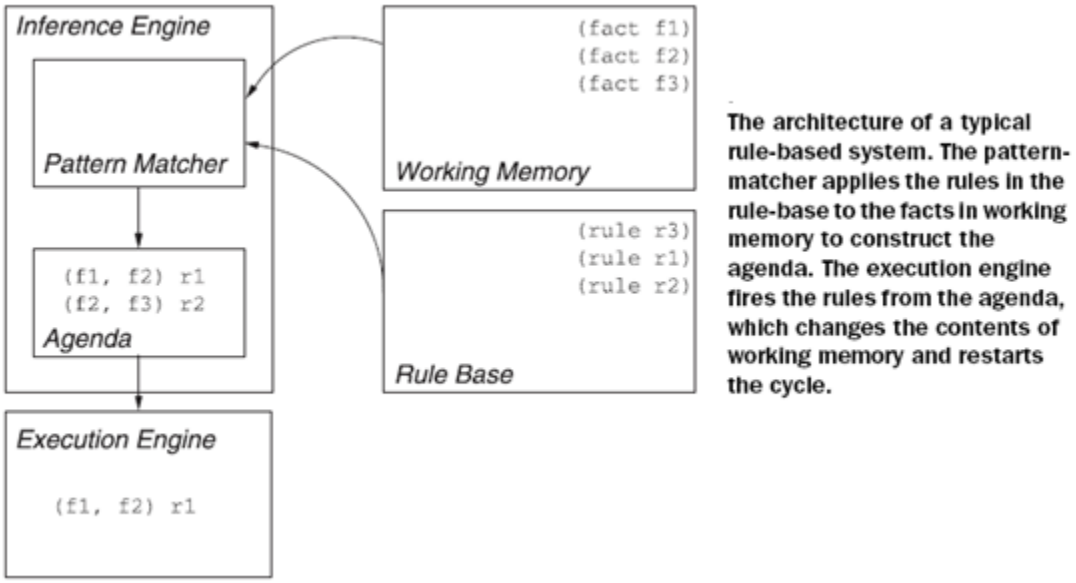


Figure 7 The inference engine

2.4.2.1 The inference engine

If you wanted to write your own rule engine, where would you start? You might begin with the most important component. The primary business of a rule engine is to apply rules to data. That makes the inference engine the central part of a rule engine.

The inference engine controls the whole process of applying the rules to the working memory to obtain the outputs of the system. Usually an inference engine works in discrete cycles that go something like this:

1 All the rules are compared to working memory (using the pattern matcher) to decide which ones should be activated during this cycle. This unordered list of activated rules, together with any other rules activated in previous cycles, is called the conflict set.

2 The conflict set is ordered to form the agenda—the list of rules whose right-hand sides will be executed, or fired. The process of ordering the agenda is called conflict resolution. The conflict resolution strategy for a given rule engine will depend on many factors, only some of which will be under the programmer's control.

3 To complete the cycle, the first rule on the agenda is fired (possibly changing the working memory) and the entire process is repeated. This repetition implies a large amount of redundant work, but many rule engines use sophisticated techniques to avoid most or all of the redundancy. In particular, results from the pattern matcher and from the agenda's conflict resolver can be preserved across cycles, so that only the essential, new work needs to be done.

Many beginning rule programmers have difficulty with the idea that the rule engine will decide the order in which the rules will be fired, but this is actually one of the great strengths of rule-based programming. The rule engine can more or less create a custom program for each situation that arises, smoothly handling combinations of inputs the programmer might not have imagined.

2.4.2.2 The rule base

The rule base contains all the rules the system knows. They may simply be stored as strings of text, but most often a rule compiler processes them into some form that the inference engine can work with more efficiently. Jess's rule compiler builds a complex, indexed data structure called a Rete network. A Rete network is a data structure that makes rule processing fast. In addition, the rule compiler may add to or rearrange the premises or conclusions of a rule, either to make it more efficient or to clarify its meaning for automatic execution. Depending on the particular rule engine, these changes may be invisible to the programmer. Some rule engines allow (or require) you to store the rule base in an external relational database, and others have an integrated rule base. Storing rules in a relational database allows to select rules to be included in a system based on criteria like date, time, and user access rights.

2.4.2.3 The working memory

We also need to store the data the rule engine will operate on. In a typical rule engine, the working memory, sometimes called the fact base, contains all the pieces of information the rule-based system is working with. The working memory can hold both the premises and the conclusions of the rules. Typically, the rule engine maintains one or more indexes, similar to those used in relational databases, to make searching the working memory a very fast operation. It's up to the designer of the rule engine to decide what kinds of things can be stored in

working memory. Some working memories can hold only objects of a specific type, and others can include, for example, Java objects.

2.4.2.4 The pattern matcher

The inference engine has to decide what rules to fire, and when. The purpose of the pattern matcher is to decide which rules apply, given the current contents of the working memory. In general, this is a hard problem. If the working memory contains thousands of facts, and each rule has two or three premises, the pattern matcher might need to search through millions of combinations of facts to find those combinations that satisfy rules. Fortunately, a lot of research has been done in this area, and very efficient ways of approaching the problem have been found. Still, for most rule-based programs, pattern matching is the most expensive part of the process. Beginning rule programmers often overlook this fact, expecting the procedural right-hand sides of their rules to represent all the computational effort in their program. Often the pattern-matching technique used by a particular rule engine will affect the kinds of rules one writes for that engine, either by limiting the possibilities or by encouraging to write rules that would be particularly efficient.

2.4.2.5 The agenda

Once the inference engine figures out which rules should be fired, it still must decide which rule to fire first. The list of rules that could potentially fire is stored on the agenda. The agenda is responsible for using the conflict strategy to decide which of the rules, out of all those that apply, have the highest priority and should be fired first. Again, this is potentially a hard problem, and each rule engine has its own approach. Commonly, the conflict strategy might take into account the specificity or complexity of each rule and the relative age of the premises in the working memory. Rules may also have specific priorities attached to them, so that certain rules are more important and always fire first.

2.4.2.6 The execution engine

Finally, once the rule engine decides what rule to fire, it has to execute that rule's action part. The execution engine is the component of a rule engine that fires the rules. In a classical production system such as MYCIN, rules could do nothing but add, remove, and modify facts in the working memory. In modern rule engines, firing a rule can have a wide range of effects. Some modern rule engines (like Jess) offer a complete programming language you can use to define what happens when a given rule fires. The execution engine then represents the environment in which this programming language executes. For some systems, the execution engine is a language interpreter; for others, it is a dispatcher that invokes compiled code.

2.5 Online Auctions

2.5.1 Agent and Multi-agent Systems

The steadily increasing interconnection of devices raises many research issues with regards to the ways in which all these distributed machines can interact effectively with other humans or machines. Artificial intelligence and software engineering have a role to play in how the computers can interact in a ”rational” way. Indeed, agent technology is a significant area of interest for such applications as telecommunications, information management, Internet search engines, electronic commerce, computer games, interactive cinema, information retrieval and filtering, user interface design, industrial process control, planning and logistics, etc. The successful adoption of this technology in all these areas will have a profound impact both on industry, and also on the way in which future computer systems will be conceptualized and implemented.

At present, there is much debate about what agent-hood is exactly, and there is nothing approaching a consensus as it is generally the case for any new field. However, an increasing number of researchers define agents as being:

1. situated or embedded in a particular environment;

2. designed to fulfil specific roles;

3. clearly identifiable entities with well-defined (and limited) resources and interfaces;

4. autonomous in the sense that they have control over their behavior;

5. capable of exhibiting flexible behavior which can be reactive, proactive, sociable or persistent.

In the context of concurrent and distributed systems, it becomes obvious that a single agent is insufficient. Many applications, if not most of them, require multiple agents, called also multi-agent systems (MAS). In such systems, knowledge, action and control are distributed among the agents, which may cooperate, compete or coexist depending on the context in which they operate. According to Weiss [89], there are two main reasons which drive forces behind the growth of the MAS paradigm in recent years.

The first is that multi-agent systems have the capacity to play a key role in current and future computer science and its application. Modern computing platforms and information environments are distributed, large, open, and heterogenous. Computers are no longer stand-alone systems, but have become

closely connected both with each other and their users. The increasing complexity of applications often lead to the design of 'individual' software agents instead of a large entity which is in general less flexible. The technology that MAS promises to provide, are among those that are urgently needed for the Internet, telecommunications, TV-web, e-commerce, e-business, etc.

The second reason is that multi-agent systems have the capacity to play an important role in developing and analyzing models and theories of interactivity in human societies.

These two reasons combined show the relevance of MAS for understanding, implementing and operating complex socio-technical systems as represented by e-business systems.

2.5.2 Auctions: From Economy to the Automatic Negotiation

Auctions are a market mechanism already introduced in the ancient world. Traditionally, they allow selling rare and unusual goods, and apply in situations where a more conventional "market", in which buyers consider the price as given, do not exist. A large informal body of knowledge on auctions has been in existence for centuries, and a more formal, game theoretic analysis of auctions began in the 1960's with the pioneering work of Vickrey [50]. The field of micro-economics and game theory that studies these and related issues is often called 'mechanism design' or 'implementation theory'. An introduction to this field can be found in the text books [51, 52].

With the widespread availability of the Internet and e-commerce technologies, economists have started to consider auctions as an important economic model. Indeed, the theory of auctions exhibits many interesting features at the practical, empirical and theoretical level. Paul Klemperer [53] gives three reasons why the theory of auctions is relevant to economists:

1. A growing number of large volume transactions are realized through auctions. Examples include the many auctions organized by governments [54], the enterprise wide electronic procurement systems for all sorts of goods, or the electronic auction markets such as EBay [55, 56].

2. Auctions offer relatively simple mechanisms in a well defined economic environment. They offer thus to economists a vast field for experimental research allowing to obtain empirical results that can be suitably validated.

3. The theory of auctions has already revealed many scientific results in economy, which have allowed to develop new methods for pricing in competitive markets. Moreover, it has also helped to further understand complex negotiation mechanisms between vendors and buyers.

Recently, we have noted a great rise in the popularity of auctions of various types [57]. This development occurred in many settings: in governments privatizations and rights allocation [58, 59]; in the many Internet auction sites [55, 56]; in the usage of techniques from auction theory for computational resource allocation [60]; in computerized agent systems [61, 62]; and trends in B2B e-commerce [63].

Besides the research on auction mechanisms undertaken in economics and operations research, the technology necessary to implement electronic auctions [64, 65, 66] is a research issue on its own. Agent technology has already been applied in different work [67, 68, 69, 70]. The application of a multi-agent paradigm to auctions can be viewed from two different points of view:

1. The mechanisms of an auction can be defined as a resource allocation problem to a set of agents. The available resources are limited with regard to the number of demands. The problem may thus be described as a market in which there are vendor agents and buyer agents which trade on items represented by the resources. These agents exhibit different acquisition capabilities which let them act differently depending on the current context or situation of the market. For example, the ”richer” an agent is, the more items it can buy, i.e. the more resources it can acquire.

2. The auctions can be viewed as a process of automatic negotiation implemented as a network of intelligent agents. Buyer and vendor agents interact in an electronic market environment to trade items. Such an approach is for example represented by AuctionBot [71]. Alternatively, auctions may form an integral part of a multi-agent architecture. An example is an architecture where supervisor agents associated with institutions offer a same product in competition with each other applying auction mechanisms in order to offer to a user agent the best price for the product.

2.5.3 Auctions for automatic negotiation

2.5.3.1 Market framework

An environment where vendors and buyers meet with the goal to sell and buy goods is commonly called a market. As there are many different interpretations of what a market is, such an environment should rather be called a market framework. Figure 8 shows an example of such a framework . There are four cases presented:

1. One vendor and one buyer directly negotiate in the classical sense;

2. Multiple vendors and one buyer are engaged in a reverse auction;

3. Multiple buyers and one vendor are engaged in a classical auction (English, Dutch, Vickery, etc.);

4. Multiple buyers and vendors trade in a market.

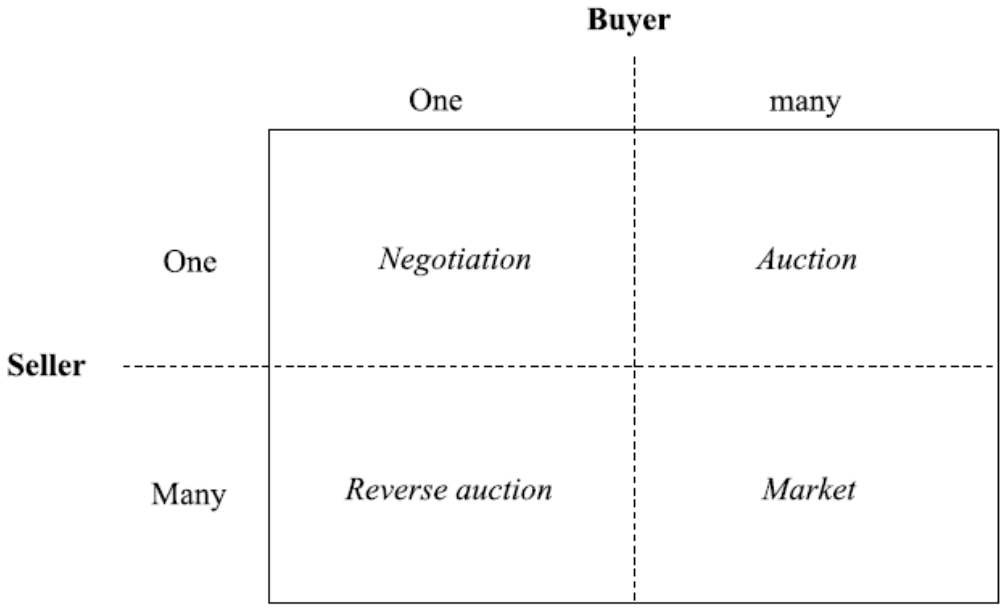


Figure 8 Market framework

The most usual form is number 3 with popular ones being the English, Dutch, first-price sealed bid and second-price sealed bid (Vickrey). In an English auction, the auctioneer begins with the lowest acceptable price and bidders are free to raise their bids successively until there are no more offers to raise the bid. The winning bidder is the one with the highest bid. The Dutch auction is the converse of the English one; the auctioneer calls for an initial high price, which is then lowered progressively until there is an offer from a bidder to claim the item. In the first priced sealed bid, each bidder submits his offer for the item independently without any knowledge of the other bids. The highest bidder gets the item and he pays a price equal to his bid amount. Finally, a Vickrey auction is similar to a first-price sealed bid auction, but the item is awarded to the highest bidder at a price equal to the second highest bid.

The distinction between negotiations, auctions and markets is, however, not so strict. This rough classification leaves therefore room for variations such as combined negotiations [72], synchronous open auctions, and combinatorial auctions. Moreover, combinations are also possible: one can imagine a model where auctions are used as a mechanism for automatic negotiation in electronic markets.

2.5.3.2 Auctions and automatic negotiation

Economic models have been adopted in various works on the problem of negotiation and resource allocation in multi-agent systems. Ferguson [73] justifies this choice by the availability of the many mathematical tools related to numerical economics. Evidently, these tools are of great value when resolving the resource allocation problem in complex information systems.

Negotiation is central to any commerce and market. It can be defined as "Mechanism that allows a recursive interaction between a principal and a respondent in the resolution of a good deal" [74].

In [61], S. Rosenchein and G. Zlotkin propose five attributes that are necessary for a 'good' negotiation mechanism: efficiency, stability, simplicity, distributivity and symmetry. These characteristics can all be found when considering an auction model for automatic negotiation:

Efficiency. To show the efficiency of an auction is often very difficult, if not impossible to undertake without imposing severe restrictions such as independent private evaluations, risk neutrality, homogeneous buyers, independent products to sell, etc. However, these restrictions are quite straightforward to implement in a multi-agent system. Indeed, the agents are often considered as rational entities seeking to maximize a well defined utility function based on precise rules. By imposing on an architecture strict social relations between the different buyers, a simple auction model can be designed, which follows the simplifying restrictions. This will assure the efficiency of the proposed auction mechanism.

Stability. An obvious method to achieve stability is to forbid an agent to cancel or reconsider offers once they have been submitted.

Simplicity. Auction mechanisms are in general quite simple to implement. This is in fact one of their strength. Considering the number of messages to communicate, the only communications necessary are the offers and the responses of the vendor.

Distributivity. At first sight, distributivity is only partially fulfilled: while there are multiple (independent) buyers, the vendor or auctioneer is a central entity. However, in a more complex environment, i.e. a market, different buyers and vendors may negotiate. In such an environment there is no global central entity anymore. The vendors may either act independently or coordinate their activities. This allows for multiple simultaneous auctions.

Symmetry. The symmetry is easily achieved by assuring that all participating agents have access in the same way to all information available to them. No agent will thus be privileged with regard to others.

There are other characteristics of auctions supporting the application of auction mechanisms for automatic negotiation. An auction restricts the negotiation variables essentially to the price and the quantity in case of multiple

items. Moreover, an open auction allows the agent to review his offers, and if the auction is public, to refine them by analyzing the offers of other participants and by considering the auction's evolution. The negotiation strategy may thus be adapted according to the rules of the market. Finally, an auction negotiates a mutually acceptable solution for both the vendor and the buyer while the market forces alone decide on the negotiation termination

The previous discussion shows that auctions are appropriate for automatic negotiations. Auctions may also be applied to efficiently design a market framework. However, some differences between negotiations and auctions with regard to electronic commerce applications can be identified [68]. The participants of an electronic auction cannot make mutual concessions or agreements. Such an auction is thus not considered as a mechanism for negotiation. This is however only true, if the negotiation is restricted to only one dimension, i.e. the current price of the item in the auction. If more dimensions such as quantity and quality are considered , auctions may be considered as negotiation mechanisms where the buyers might make concessions on one or the other dimension in order to favor one specific dimension. Dually, the vendor might operate on the different dimensions and dynamically adjust the selling strategy in the auction.

3 Literature Review

3.1 Trust management

Trust management using reputation models are based on prior history of users and/or feedback gathered from other entities. Shmatikov and Talcott proposed a formal model that precisely defined the notion of reputation, and can be used to reason about trust [6]. Based on the license-based digital rights language, they used licenses to formalize both “good” and “bad” behaviors, which specify obligations and forbidden actions, respectively. Selcuk and his colleagues proposed a reputation based trust management protocol for P2P networks, where users rate the reliability of other parties, and share this information with their peers [7]. Recently, reputation based trust management has been applied to sensor networks. For example, Ganeriwal and Srivastava proposed a reputation-based framework for sensor networks (RFSN), which allows development of a community of trustworthy sensor nodes based on their behaviors, and can maintain the reputation for sensor nodes and evaluate their trustworthiness [8].

Trust and reputation management has been a promising approach to building trustworthiness in networked systems. However, many reputation models suffered from a major drawback – there is no effective mechanism to prevent users from giving false information when making a recommendation. In addition, this approach fails to evaluate the trustworthiness of new users or those who have not yet received enough reputation feedback. As a result, reputation based trust management models are usually *not* sufficient to ensure the trustworthiness of a system.

On the other hand, policy-based trust relies on trusted certification authorities (CA) and signed certificates as well as access control policies to determine whether a requester is trusted or should not be allowed to access a certain resource [38]. Declarative policy rules are well suitable for specifying access control conditions for access permissions to certain resources. Policy languages such as REFEREE [39] and KeyNote [40] can support authorizing the trustee automatically by determining whether certain credentials are sufficient for performing a certain action or accessing a certain resource. At the authentication level, trust is bound to a certain identity or membership, which is not changeable during a period of time. The focus of such an approach is on credential matching policy.

However, no research has yet emerged for updating the level of trust based on evidence of actual behaviors in real-time [41]. In some recent work, Bonatti and his colleagues proposed an integrated trust management framework that combines rule-based and credential based trust, which is capable of addressing the complexity and the variety of semantic web scenarios with both structured organizational environments and unstructured user communities [38]. Most of the existing work on trust management emphasized defining static mechanisms or algorithms to calculate users' trustworthiness value. The evolution of trust has been seldom discussed in detail in practical contexts due to the difficulty in reconfiguring the trust management model based on newly acquired evidence [41]. In contrast, we propose an agent-based trust management (ATM) module that facilitates real-time trust re-evaluation by not remaining in the initial trust result but watching all the auctions progress and finding shilling behaviours. As a result, our approach provides a better foundation towards building a trustworthy networked system. We introduce our module in the context of online auction systems, which require strong trustworthiness of the auction house with true and timely evaluation of users' bidding activities. Previous trust management models only consist of a security agent that is responsible for all tasks of shill detection, which is *not* scalable when the number of users increases dramatically. To solve this problem, in our work, we define the ATM module as a multi-agent system where each client is responsible for finding shilling behaviours on its own. In order to efficiently detect shill bidders, monitoring agents can run concurrently and use real-time auction data to search for shill suspects based on patterns of shilling behaviors. When a shill suspect is detected, the agent can use available user information and auction data to verify whether the shill suspect is an actual shill. Furthermore, unlike previous work, the ATM module introduced in this paper is a generic module that can be applied in both agent-based online auction systems and conventional online auction houses such as eBay and Yahoo!Auctions.

Two models that really affected our approach are further analyzed in the following paragraphs.

3.1.1 Abdul-Rahman and Hailes Reputation Model

Abdul-Rahman and Hailes [43] have proposed a reputation model which draws in part from prior work defining trust in a computational manner [46]. Their main objective is to allow trust decisions based on a variable scale, not the binary decision usually employed by cryptographers, but more a subjective degree of belief by which agents make choices.

Their working definition of reputation relies on the experiences of the agent themselves, part of which may be information gathered from other agents. This allows them to generalize reputation information from the agent and also the agents' neighbors.

In their definition, an experience is either information gathered in a personal experience or reliance on a recommendation from another agent. The first is known as *direct trust* while the second is referred to as *recommender trust.* Direct trust is associated with agent's name, a context, and a degree of trust, which is defined as one of the following: Very Trustworthy, Trustworthy, Untrustworthy, or Very Untrustworthy. Evaluation of recommender information is defined in a similar way.

The agent then is able to keep a set *Q* of 4-tuples of information related to their own experiences, where *Q* is a set of cross products of contexts, agents, and subjective reputation ratings. Likewise, the agent keeps a set *R* of information related to recommenders. The algorithm allows the agent to keep adjustment values on recommenders which do not see the world as they do (i.e. the recommender's "very trustworthy" is only a "trustworthy" to the agent). This set R consists of cross products of contexts, agents, and adjusted recommendation information. Information from *Q* and *R* are combined using a statistical formula, and a single evaluation of trust is obtained.

Abdul-Rahman and Hailes account for a number of the design concerns associated with reputation systems. They provide for context, they consider that not all recommender information will be reliable, so they allow adjustments when evaluating it. Also, they allow a means of bootstrapping new agents into the network, by providing them with a number of pre-trusted entities to provide them with recommender information as they build their own set of experiences.

3.1.2 Mui, Mohtashemi, and Halberstadt's Reputation Model

Mui, Mohtashemi, and Halberstadt [44, 45] provide a rather different notion of reputation than ARH. Their idea is based on the concept of reciprocity and its role in cooperation as described by social scientists. Additionally, they differentiate between trust and reputation, since reputation is one measure whereby trust can be gained, i.e. a good reputation is not always sufficient to inspire trust. The following diagram depicts the relationship that Mui et.al. see between reputation, trust, and reciprocity.

The following definitions are given in the paper for these three concepts:

• **Reciprocity:** mutual exchange of deeds (such as favor or revenge)

• **Reputation:** perception that an agent creates through past actions about its intentions or norms.

• **Trust:** a subjective expectation an agent has about another's future behavior based on the history of their encounters.

Mathematically, this system is represented using Bayesian probability theory. Each agent keeps track of a set of encounters, along with their results. Reputation is a probability measure from 0 to 1 on the likelihood of another agent

to reciprocate. Reputation scores are kept for each pair of agents in the model. Observed encounters count equally with experienced encounters, and are kept in the same set of encounter data. Trust, τ_{ab}, is defined as an agent's *a* expectation for reciprocation by a particular agent *b* in a given context. Agent *a* uses its encounter history with agent *b* to calculate a reputation $\hat{\theta}_{ab}$ using a Beta prior distribution. This process is as follows:

The encounter history D_{ab} is the set of individual encounters $\{x_{ab}(1), x_{ab}(2), \ldots, x_{ab}(n)\}$. The prior probability uses a Beta distribution function, $p(\hat{\theta})=Beta\ (c_1\ ,\ c_2)$ where c_1 and c_2 are parameters determined by prior assumptions such that $0 \le c_1 \le 1$ and $0 \le c_2 \le 1$. An estimator for reputation is the proportion of successful encounters *p* to total encounters *n,* so $\hat{\theta}_{ab}=\frac{p}{n}$

Assuming that the probability of successful encounters is independent, the likelihood of *p* cooperations in *n* attempts (probability of cooperation) can be modeled as

$$L(D_{ab}|\hat{\theta}) = \hat{\theta}^{p}\ (1-\hat{\theta})^{\ n-\ p}$$

After combining the prior and the likelihood, we get a posterior estimate of

$$p(\hat{\theta}\ |D_{ab})=Beta\ (c_1+p\ ,\ c_2+n-p)$$

Trust τ_{ab} is defined as the probability of cooperation on encounter *n+1*, or the probability that $x_{ab}\ (n+1)=1$. This means that at *a's* next encounter with *b*, agent *a* will estimate the probability of cooperation from agent *b* to be

$$\tau_{ab} = p(\ x_{ab}\ (n+1)=1|D_{ab})=\frac{c_1+p}{c_1+c_2+n}$$

Mui's model accounts for different levels of confidence, however, it does not differentiate between observed encounters and real encounters, assuming that all observations are first hand and not transferred by "word of mouth." For the case in which two unknown agents do not have any encounter history, Mui chooses a uniform distribution with $c_1=1$ and $c_2=1$, so that τ_{ab} =0.5 initially.

In [82], Wolfgang et all have discussed the PeerTrust policy language and how to use it for negotiating and establishing trust in a distributed elearning environment. The limitations of their work is that there are no formal guarantees that trust negotiations will always terminate and will succeed. Our system follows the FIPA Specifications for the negotiations between the agents and although the trust of an agent is also monitored during the negotiation we always guarantee that the negotiation will terminate. Moreover a single resource and a single type of access to that resource but agents in our platform have different

level of access according to their trust level and they can participate in different auctions that have different access levels.

In [83] Almendra et al explore the idea of trust delegation, i.e. using trust information from other agents. Nonetheless, their proposed methods not adequate to express some conditions, e.g. those based on quantity constraints and on statistical calculations. The problems are the result of expressiveness of the language they use but we solve them with the use of JESS.

In [86] Golbeck et al offer an approach to calculate trust on a web-based social network, grounded on the concept of trust as reliance among agents in the network. Each agent states trust on some other agents; this leads to a social network whose directed edges express trust of an agent on other. This trust attitude has degrees, which are weights of the graph edges. From this model, Golbeck proposes algorithms to infer trust among people connected only indirectly through the network. We adopt a different approach to trust measurement, as we believe that assigning numbers to trust relationships and then making computation with these numbers can be misleading, as the semantics of the result is not clear, even if it obeys some kind of intuition about how trust works in real life. We use the idea of trust as reliance among agents but only direct reliance not reliance through others.

In [84] Haiping Xu et al use an agent-based trust management (ATM) module to monitor, analyze and detect shill bidders dynamically. They plan to build a bidder's intention model and make decisions on detection of actual shills with a Bayesian network learned from historical auction data and information about users. Their work focuses on the auctioneers side and all the monitoring is done from the Auction house's side but our work does this work form the clients's side allowing him to be independent and so he doesn't have to base his actions on the Auction house. Moreover in their work the Clients taking part are humans and no intelligent agents as in our work and the negotiation level is not automated so this set many limits in their efford to detect shilling behavior. Our System builds a bidder's intention model that takes shill decisions by monitoring, analyzing and detecting shill bidders dynamically but also taking under consideration bidder's previous information and data about the auction.

In [85] Uzun et al describe a protocol to distinguish the malicious responses from benign ones by using the reputation of the peers providing them. Since in P2P networks a central server is typically not available, their protocol relies on the P2P infrastructure to obtain the necessary reputation information when it is not locally available at the querying peer. But their protocol does not distinguish between malicious peers and the peers that spread that do malicious actions due to their carelessness, which we believe is the right way to deal with careless peers from a practical point of view. In our work the agent's strategy implementation is designed in such a way that there is no possibility an agent will accidentally do shilling. If it is sure that there is a better offer for the exact same item the CL-agent stops bidding.

3.2 Negotiation

At this moment, many research works about automated negotiation are being developed. The model proposed by Faratin [Faratin et al. 1998] is consistent and really near of the idea to automate negotiations in the e-commerce. For example, it is already possible to give to negotiating agents the capacity to make decisions in an independent way. Moreover, the idea of introducing negotiation capacity is another advance.

In the other hand, some important cognitive aspects linked to negotiation activities have not been boarded yet. The first one is the ”reflection capacity”, because in the end of each negotiation it would be important to a negotiator to evaluate how efficient was the interaction with his opponent, identifying the positive points and the negative ones. After that, another important characteristic would be ”EMPATHY”, what means the attempt to understand the real necessities or preferences of his opponent, in order to decrease the divergences between them. And at last, add to the agent the capacity to do many negotiations at the same time, enabling the agent to have a real vision about the negotiation, because is supposed that he faces different opponents who have different proposals. Another factor to be considered is the decision to send an alternative product that could have an individual and automated connotation. Actually, the decision to send an alternative product does not consider the aspects of the opponent. There is no mechanism to send or to displace a product in a current negotiation for a more acceptable one.

Another aspect to be considered is the fact that the substitution of a product in the negotiation is restricted only for another of the same kind, but not with different specifications. In a negotiation, it would be interesting to substitute a product that is being negotiated for another one that satisfies the interests of the two participants, even with different specifications. The method of evaluation of the proposals has a subtle problem: it restricts a function in just one attribute of the negotiation. This is not interesting, because a criterion may be connected to more than one attribute, or the opposite.

The works proposed by [76, 77] introduced important advances in the topic of Automated Negotiation in e-commerce, however some points must be improved. First, [77] proposes an open environment for automated negotiation. This environment is based on mechanisms of central negotiation which controls the functioning of a negotiation environment and also the necessary conditions to realize agreement. This is a problem because the circumstances of the

negotiation that both agents have to face to make a deal may have different conditions and extremely variable aspects.

The work proposed by [77] has no support for agents. In fact, the designers need to know the rules for negotiation and implement them directly in their agents for each possible negotiation mechanism. The conditions of an agreement are all predefined and there is only one negotiation object involved which is previously specified. So, the bargaining power between the two agents is reduced, because it will not be possible to negotiate the specifications of the products on different negotiation conditions.

Bartolini [76] tries to overcome this problem connecting a definitive agent to a specific business domain using ontology. Bartolini [76] proposes that only one ontology may be responsible for all the protocols of negotiation, so when one agent joins one electronic market it will be able to negotiate with no need to make some type of previous adjustment. The first problem is that if an electronic market has a different negotiation protocol the agent will not be able to negotiate, because the protocol was not implemented. The second problem is that some business domains need a bigger expressive power with many different terms and intentions, what may turn the ontology unable due to the big amount of terms and subjects.

In the work of Faratin [78], there is an attempt ”to understand” the opponent and offer proposals that are good for both parts. Moreover, a model of negotiation was simplified in order to deal with agents of any mechanism of negotiation. The result of this work is satisfactory when the set of ”opponents”, which the agent will have to deal with, is homogeneous and previously well described. Therefore, individual proposals for unknown environments are not satisfactory. Besides, if it is necessary the interaction with dynamic environments where the preferences of negotiating agents may change, this model is not able to solve the difficulties. So, mechanisms that can adapt to the environment became extremely necessary.

In [80] Bădică, et al conceptualized agent negotiations as consisting of: (i) negotiation participants and a host where the negotiations take place; (ii) a generic negotiation protocol, and (iii) a taxonomy of rules for enforcing a specific negotiation mechanism. They provide implementation details using JADE and JESS and by presenting a scenario involving multiple English auctions performed in parallel. In particular, in our implementation, the rule-based sub-agents of the negotiation host share a single JESS rule engine, rather than having separate rule engines within each sub-agent.

They suggest that they need to assess the generality of their implementation by extending it to include other price negotiation mechanisms as the simple scenario they use is sufficient for the purpose of their paper, i.e. to illustrate how multiple rule-based automated negotiations can be performed in parallel but it has the disadvantage that the same client participates as active

member in many auctions at the same time having the possibility to win in more than one auction.

In our system there is no such possibility according to the strategy we follow and our scenario makes the agents follow different mechanisms according to the parameters they take under consideration.

In [81] Lochner et al describe a scripting language which is one important facility of their configurable auction server, AB3D. Although the system as a whole remains a work in progress (e.g., we are currently working to extend it to support multiattribute auctions), they have implemented and used several mechanisms specified by AB3D scripts, for example the three auction types employed in the TAC Classic travel market [18], so their work covers a wide range of negotiation issues but the system as a whole remains a work in progress and is not fully operational.

4 System's architecture

This section describes our platform in detail. It starts with a layered model of the system's structure, presents its basic components, explaining their features and, finally, analyzes the platform's architecture, along with examples of interaction.

4.1 Design

In describing the structure of our platform, we take a layered approach. The platform's basic component diagram is shown in Figure 9. The platform integrates two preexisting technologies; JADE [33] for configuring the peer-to-peer network and planning and managing the multi-agent character of the system, JESS [34] for the declarative rules used in the trust level and for finding clients that try to raise price of an item. Details about these systems are further discussed in the following sections. For analyzing the items and auctions ontologies we used the owl-API. No distinction is being made between the customer and the auctioneer roles in the diagram, because the system components support the same operations and information flows regardless of the user's role.

The top layer, called *platform layer*, encapsulates the essential mechanisms for controlling the *basic service modules* and managing their inputs and outputs, in order to receive, process, filter and forward information among them. It provides core functions and unites the basic modules of the platform.

Under the platform layer the three basic modules are defined. These modules are not interconnected, but communicate through the platform layer instead. JADE is responsible for implementing protocols to allow the application to function as a peer, collaborate with other peers and deploy peer-to-peer services. It also takes the role of automating the negotiation procedure of auctions by creating, controlling and monitoring software agents that represent human users. Furthermore, the Semantic Analysis facilitates semantic publish and discovery of products on the network.

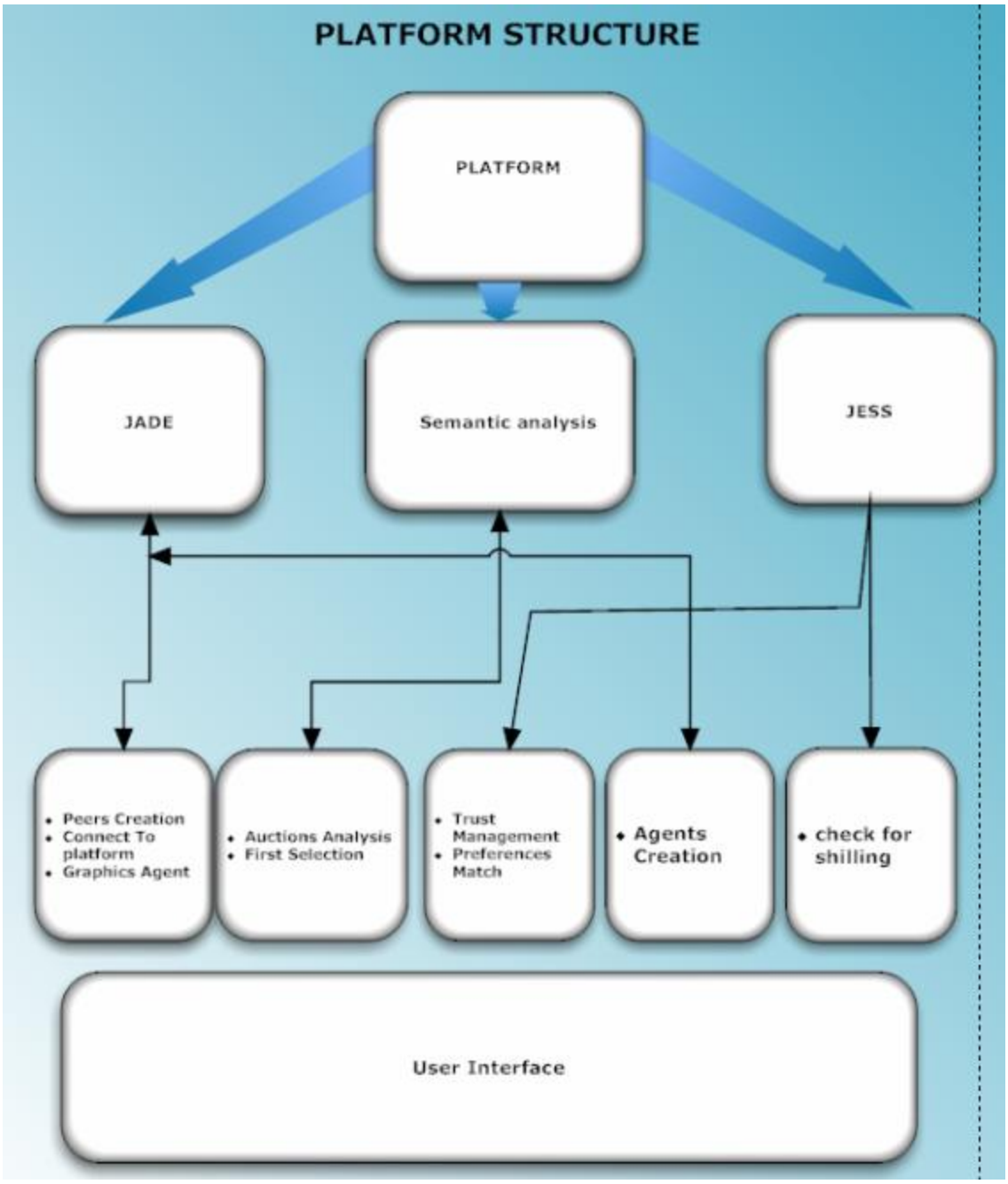


Figure 9 The platforms Layers

At the bottom layer, the user interface is defined, which contains the group of modules that work together to provide an integrated user interaction with the system. It offers graphical control over the platform's functions and presents responsive information and graphical charts about the progress of the user's running auctions.

The following example illustrates a general interaction between the aforementioned elements. A human user participates in the platform environment by initiating the application, either as a bidder or as a supplier. The platform layer

activates the basic components and presents the UI main frame. It also instructs JADE to submit an admission request for joining the network and obtaining an identity as a new room. Acting as auctioneer, the user publishes an OWL ontology describing the selling products and their auction specifications while the system contacts the JADE agent platform, with the intervention of the corresponding middleware, to create an agent to govern the auction. As a customer, the user searches the network for products. He can choose the domain of the items he wants and then see from the list of results all the items details and choose which ones he is interested in. Once the desired items are found the customer's agent gather information about the auctioneers and sets their level of trust. The agent also gets users preferences and calculates their matching result with his choices. Then the JADE middleware constructs the agents that will participate in the auctions on behalf of the user and directs them to the appropriate auction places. The interface updates information in real time to inform the user about the progress of all running sessions.

4.1.1 JADE

For the development of our negotiating agent architecture we chose JADE [35], [33]. According to [36], 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.10 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 10), different containers in the same platform (e.g. A1 and A2) or different platforms (e.g. A4 and A5).

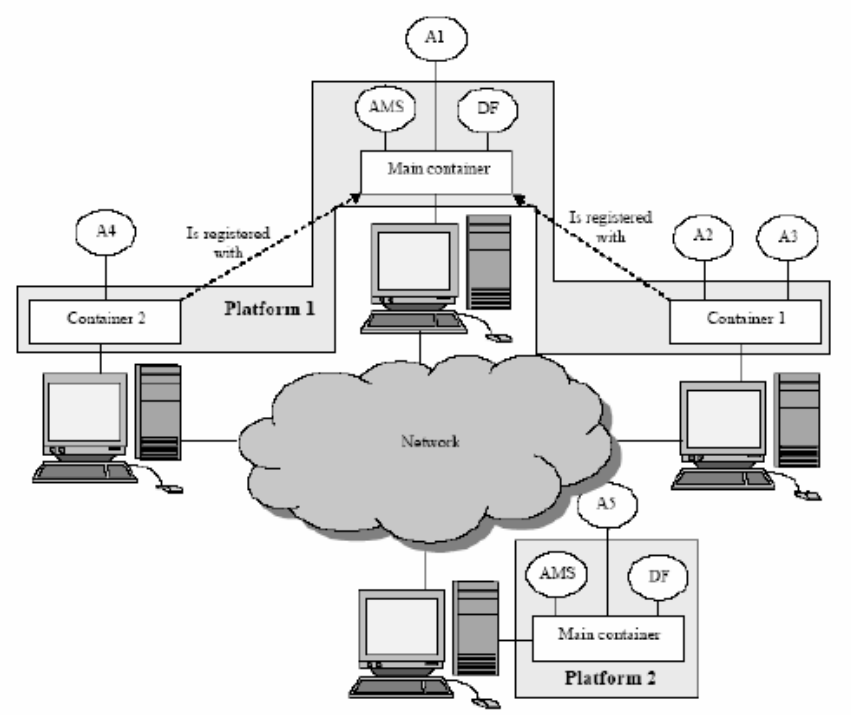


Figure 10 JADE platform example

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 [37] 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.

4.1.2 Jess

Jess is a rule engine and scripting environment written entirely in Sun's Java language by Ernest Friedman-Hill at Sandia National Laboratories in Livermore, CA. Using Jess, Java software can be built that has the capacity to "reason" using knowledge that is supplied in the form of declarative rules. Jess is small, light, and one of the fastest rule engines available. Its powerful scripting language gives access to all of Java's APIs.

Jess, or the Java Expert System Shell as it was first known, is rather more than the name suggests, comprising a general-purpose programming language which can access all Java classes and libraries. Mainly, Jess is a rule engine for the Java platform and in order for it to function; some logic must be specified in the form of rules using one of two formats: the Jess rule language or XML. When the rule engine runs, the rules are carried out. Rules can create new data, or they can do anything that the Java programming language can do. Although Jess can run as a standalone program, usually the Jess library is embedded in Java code and is manipulated it using its own Java API or the basic facilities offered by the javax.rules API.

Jess uses an enhanced version of the Rete algorithm to process rules. Rete is a very efficient mechanism for solving the difficult many-to-many matching problem 0. A typical rule-based program has a fixed set of rules while the working memory changes continuously. However, it is an empirical fact that, in most rule-based programs, much of the working memory is also fairly fixed from one rule operation to the next. Although new facts arrive and old ones are

removed at all times, the percentage of facts that change per unit time is generally fairly small. For this reason, the obvious implementation for the rule engine is very inefficient. In the Rete algorithm, the inefficiency described above is alleviated by remembering past test results across iterations of the rule loop. Only new facts are tested against any rule LHSs. Additionally, new facts are tested against only the rule (their left-hand-sides) LHSs to which they are most likely to be relevant.

Jess has many unique features including backwards chaining and working memory queries, and of course Jess can directly manipulate and reason about Java objects. Jess is also a powerful Java scripting environment, from which can create Java objects, call Java methods, and implement Java interfaces without compiling any Java code.

A Jess rule is something like an “if...then” statement in a procedural language, but it is not used in a procedural way. While “if...then” statements are executed at a specific time and in a specific order, according to how the programmer writes them, Jess rules are executed whenever their “if” parts (their left-hand-sides or LHSs) are satisfied, given only that the rule engine is running. This makes Jess rules less deterministic than a typical procedural program.

```
Jess> (defrule welcome-toddlers
"Give a special greeting to young children"
(person {age < 3})
=>
(printout t "Hello, little one!" crlf))
```

The above rule has two parts, separated by the "=>" symbol (which can be read as "then"). The first part consists of the LHS pattern (person {age < 3}). The second part consists of the RHS action, the call to println. The LHS of a rule consists of patterns which are used to match facts in the working memory, while the RHS contains function calls.

Jess can be used in many ways. Besides the different categories of problems Jess can be applied to being a library and used in many different kinds of Java programs. Jess can be used in command-line applications, GUI applications, servlets, and applets. The reactive rules used for catching user’s deviation and the obstacle report were written in Jess language and triggered by Jess rule engine.

4.1.3. Ontology Web Language (OWL)

The Web Ontology Language (OWL) is a family of knowledge representation languages for authoring ontologies, and is endorsed by the World Wide Web Consortium. OWL is intended to be used when the information contained in documents needs to be processed by applications, as opposed to situations where the content only needs to be presented to humans **Error! eference source not found.**. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. OWL has more facilities for expressing meaning and semantics than XML, RDF, and RDF-S, and thus OWL goes beyond these languages in its ability to represent machine interpretable content on the Web. OWL is a revision of the DAML+OIL web ontology language incorporating lessons learned from the design and application of DAML+OIL. OWL provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users **Error! Reference source not found.**.

OWL Lite supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and OWL Lite provides a quick migration path for thesauri and other taxonomies. Owl Lite also has a lower formal complexity than OWL DL.

OWL DL supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class). OWL DL is so named due to its correspondence with **description logics**, a field of research that has studied the logics that form the formal foundation of OWL.

OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

Each of these sublanguages is an extension of its simpler predecessor, both in what can be legally expressed and in what can be validly concluded. The following set of relations hold. Their inverses do not.

Every legal OWL Lite ontology is a legal OWL DL ontology.

Every legal OWL DL ontology is a legal OWL Full ontology.

Every valid OWL Lite conclusion is a valid OWL DL conclusion.

Every valid OWL DL conclusion is a valid OWL Full conclusion.

For the needs of the current thesis the language selected is OWL-dl for the reasons described in the following paragraph.

First of all the creation of disjointness of classes which guarantees that an individual that is a member of one class cannot simultaneously be an instance of a specified other class. We used it to show that the classes Statue and Painting are disjoint. Moreover our platform is designed to work for every domain and not only the Artifacts domain we used so with the use of the OWL-API all the graphics concerning the representation and retrieval of information concerning the domain are dynamically designed. This results in the ability to be able to use any domain one can imagine and it can be represented using a more complex OWL ontology but the platform will not face any difficulty or problem when trying to get the information it needs from it. In order to be able to use any domain without difficulty we also used a Protégé plug in (called **beangenerator**) implemented by C.J. van Aart from Department of Social Science Informatics (SWI) University of Amsterdam, it is possible to define the ontology using Protégé and then let the beangenerator automatically create the ontology definition class and the predicates, agent actions and concepts classes. This process is very convenient as it allows working with an ad-hoc graphical tool (as Protégé is) instead of writing Java code in the ontology definition process so the java beans for any OWL ontology can be generated automatically. This add on is one more reason we used OWL as the equivalent plug in for RDF creates java beans with many errors that cannot be used directly.

4.1.4. Protégé 3.4 beta

Protégé 3.4 beta is the platform used to design the User Tracking & Location Modeling ontology. Protégé is a free, open-source platform that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies 0. At its core, Protégé implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. Protégé can be customized to provide domain-friendly support for creating knowledge models and entering data. Furthermore, Protégé can be extended by way of a plug-in architecture and a Java-based Application

Programming Interface (API) for building knowledge-based tools and applications 0.

The Protégé platform supports two main ways of modelling ontologies - frame-based and OWL, each with its own user interface. The Protégé-OWL editor is an extension of Protégé that supports the Web Ontology Language (OWL), which is selected to develop our ontology. An important difference between Protégé and OWL is that OWL does not use the Unique Name Assumption (UNA). This means that two different names could actually refer to the same individual.

Protégé is a flexible, configurable platform for the development of arbitrary model-driven applications and components. Protégé has an open architecture that allows programmers to integrate plug-ins, which can appear as separate tabs, specific user interface components (widgets), or perform any other task on the current model. The Protégé-OWL editor provides many editing and browsing facilities for OWL models, and therefore can serve as an attractive starting point for rapid application development. Developers can initially wrap their components into a Protégé tab widget and later extract them to distribute them as part of a stand-alone application.

In addition to providing an API that facilitates programmatic exploration and editing of OWL ontologies, Protégé-OWL features a reasoning API, which can be used to access an external DIG compliant reasoner, thereby enabling inferences to be made about classes and individuals in an ontology.

A DIG (DL Implementation Group) compliant reasoner is a Description Logic reasoner that provides a standard access interface (a.k.a. the DIG interface), which enables the reasoner to be accessed over HTTP, using the DIG langauge. The DIG language is an XML based representation of ontological entities such as classes, properties, and individuals, and also axioms such as subclass axioms, disjoint axioms, and equivalent class axioms. The DIG langauge contains constructs that allow clients to "tell" a reasoner about an ontology and also "ask" a reasoner about what it has inferred, such as subclass releationships.

It is generally admitted that Protégé is not bug-free, but there is a lively mailing list which can help resolve issues and to which anyone can also submit bug reports or improvement suggestions.

4.1.5. Protégé-OWL API

The Protégé-OWL API is an open-source Java library for the Web Ontology Language (OWL) and RDF(S). The API provides classes and methods

to load and save OWL files, to query and manipulate OWL data models, and to perform reasoning based on Description Logic engines 0. Furthermore, the API is optimized for the implementation of graphical user interfaces.The API is designed to be used in two contexts:

- For the development of components that are executed inside of the Protégé-OWL editor's user interface.
- For the development of stand-alone applications (e.g., Swing applications, Servlets, or Eclipse plug-ins).

The Protégé-OWL API is centered around a collection of Java interfaces from the model package. These interfaces provide access to the OWL model and its elements like classes, properties, and individuals. Application developers should not access the implementation of these interfaces (such as DefaultRDFIndividual) directly, but only operate on the interfaces. Using these interfaces programmer doesn't have to worry about the internal details of how Protégé stores ontologies. Everything is abstracted into interfaces and the code should not make any assumptions about the specific implementation. The most important model interface is OWLModel, which provides access to the top-level container of the resources in the ontology. OWLModel can be used to create, query, delete resources of various types and return objects for specific operations.

OWL and RDF resources are globally identified by their URIs, such as http://www.owl-ontologies.com/travel.owl#Destination. However, since URIs are long and often inconvenient to handle, the primary access and identification mechanism for ontological resources in the Protégé-OWL API is their name. A name is a short form consisting of local name and an optional prefix. Prefixes are typically defined in the ontology to abbreviate names of imported resources. The Protégé-OWL API makes a clear distinction between named classes and anonymous classes. Named classes are used to create individuals, while anonymous classes are used to specify logical characteristics (restrictions) of named classes.

4.2 The Platform

4.2.1 Semantic Character

Traditional Web-based product searching based on keywords searching seems insufficient and inefficient in the 'sea' of information. Ontologies have shown to be the right answer to knowledge structuring and different approaches are presented for modeling and sharing a particular domain by a group of people. Until now, systems based on centralized ontology schemes suffer from difficulties concerning development and maintenance. Our infrastructure takes advantage of local ontologies, allowing participants to build and maintain their own OWL ontologies for describing products for sale. Every retailer who wishes to trade one or more products on the network builds an OWL ontology following the conventions of the public OWL Schema for the particular domain of the corresponding product, stores it locally and publishes its location, so that other members of the network can view its descriptions.

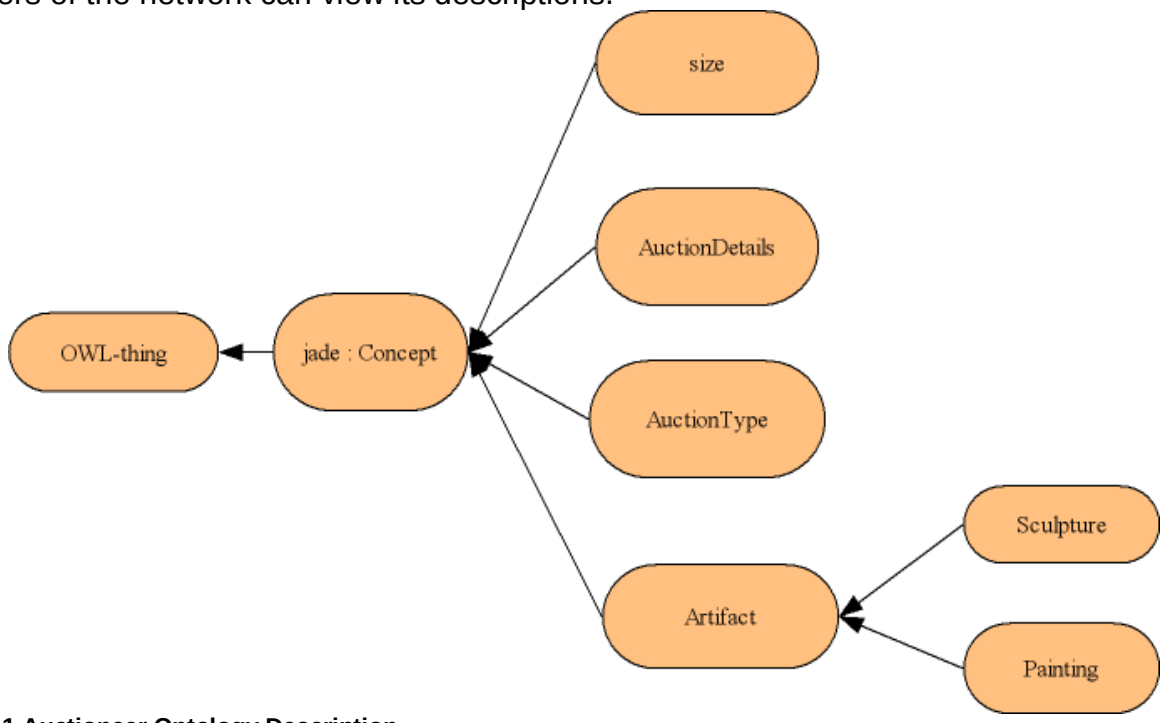


Figure 11 Auctioneer Ontology Description

In our project we have developed an ontology which has a part that is specifically about auction-related concepts and relations, and a part that represents product descriptions and enriches them with valuable metadata to better describe their features. The former part serves transactional needs, while the latter covers informational needs for product specifications.

More specifically, each item auctioned should be related with a specific domain. A retailer describes metadata about products in the corresponding domain ontology of that product. An example of such an ontology concerning the

domain of artifacts is shown in figure 12. Similar schema descriptions can be structured for other product domains, such as books, tickets, clothing etc.

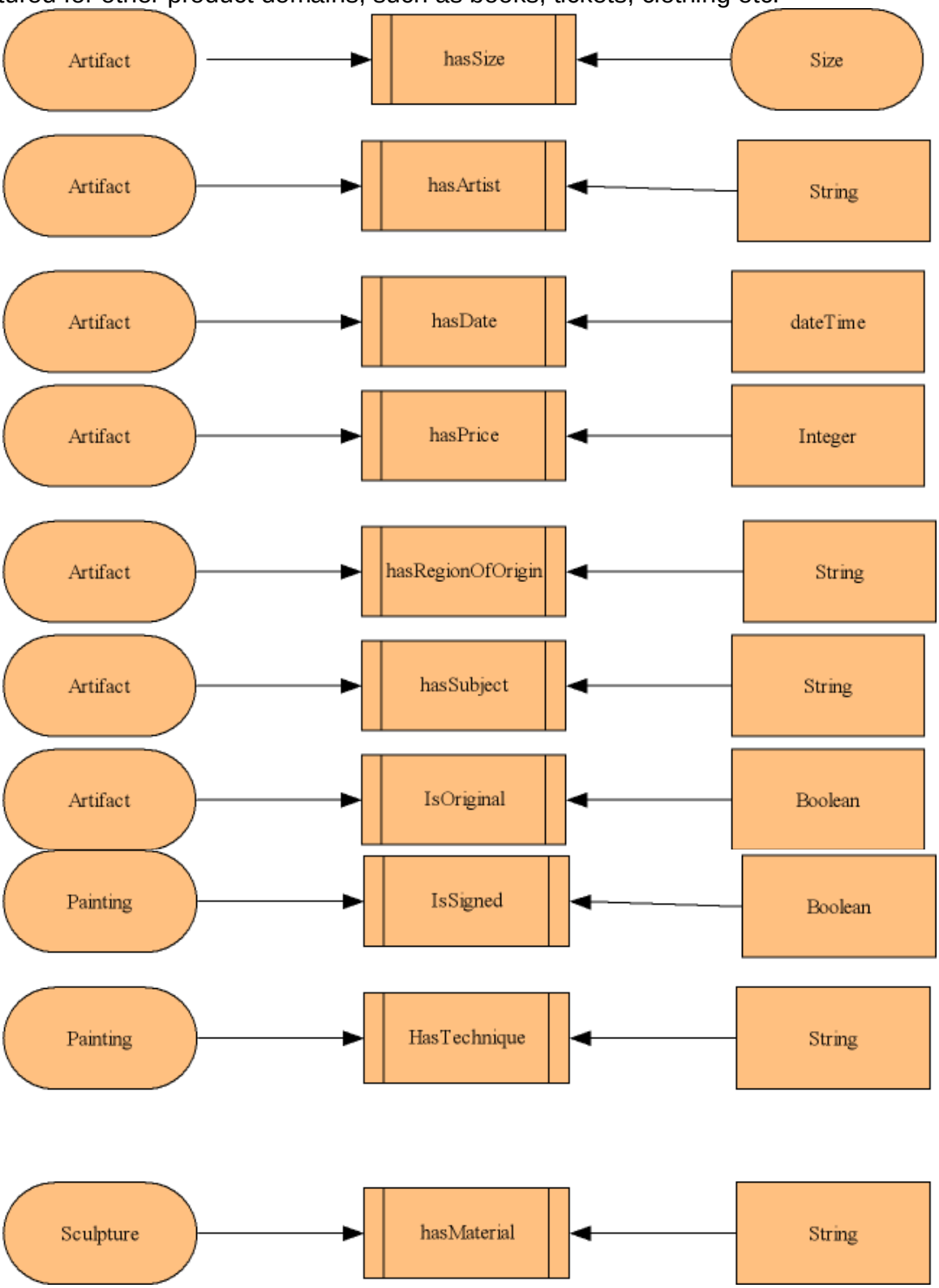


Figure 12 Artifact Concept Relationships

The auction ontology, on the other hand, captures the characteristics of a particular auction session combining knowledge from auction protocols and other common trading concepts to specify the context in which the system operates. Roughly speaking, it is used to model all information needed for an auctioneer to initiate a new auction session and for a customer to determine a desired session based on criteria, such as the broker's identity, starting and ending time etc. Figure 13 presents a representation of the auction OWL Schema.

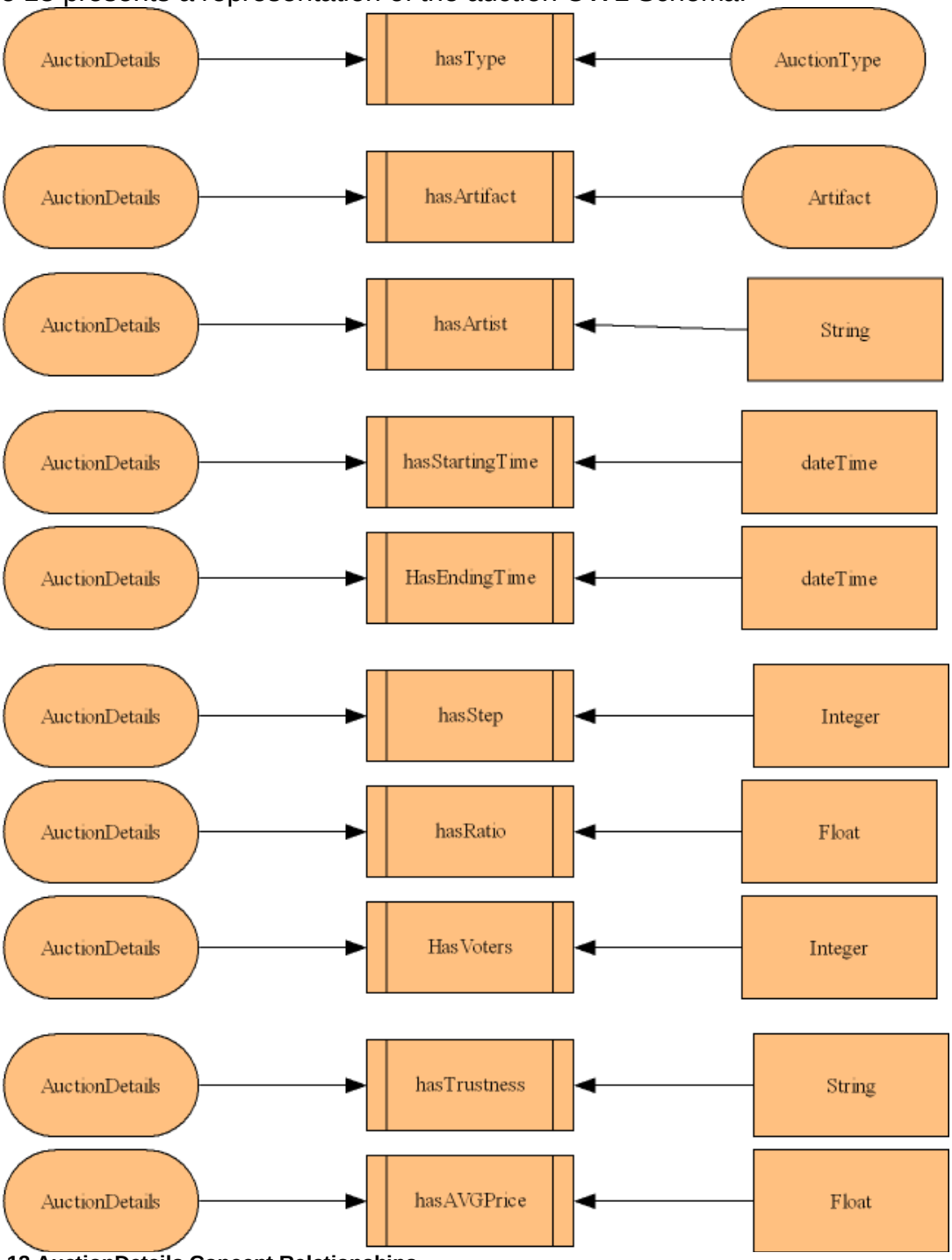


Figure 13 AuctionDetails Concept Relationships

4.2.2 Multi-Agent Character

Agent technology represents a potentially novel way of conceptualizing and implementing e-Commerce transactions. The capability of agents to offer, among others, automation in job delegation and execution, coordination and advanced communication with other agents, mobility and monitoring is exploited in our platform to reduce the tremendous time and human resources invested in on-line trading. Specifically, agents are used to facilitate the connection of buyers and sellers and to automate the process of negotiation in the context of auction scenarios. Users may decide to participate in multiple auctions at the same time, when the result of one auction may affect the action taken for the other. Agents automate bidding actions and make inferences for determining the optimum path, when interrelated auctions are involved, based on the human user's preferences and on their local knowledge. The platform currently supports two types of auctions; English and Vickrey.

We recognize four basic types of agents operating in the platform: G-,A-, C- and CL-agents

Auctioneer Agent (A-agent)

The A-agent is the auctioneer's representative in the platform's network. It manages and coordinates the execution of a specific auction and is responsible for the enforcement of rules governing the negotiation among all conversing parties. Upon initialization, the agent accepts its user's preferences concerning the auction it will conduct, such as the type of the auction, it's code name, the start and end time, the reserve price, the bid increment value, etc. This information is captured in the product's domain ontology that the user has published on the network. As the auction progresses, the A-agent contacts both the participant agents and the G-Agent to exchange data via a standard set of messages, whose format is specified by the negotiation interface (see also section). Finally, the A-agent declares auction termination and announces winning offers according to the negotiation rules.

Customer Agent (C-agent)

Customers may initiate one or more auction sessions, participating concurrently in one or more auctions in each of them. Each session has one coordinator agent, the C-agent, whose role is to manage the distinct sub-tasks that a session is decomposed into, which involve the different auctions that the customer has selected to bid in. This agent represents the user's intelligent interface to the platform because it performs the necessary actions to achieve the goal of purchasing the desired product with the best to its owner profit among all auctions that it monitors and it also checks for shill behaviour. The C-agent controls the allocation of bids across the auctions, relying on information about their progress and on its internal strategy for pursuing and maintaining its goal, but does not participate in any of them directly.

Clone Agents (CL-agents)

The CL-agents are the actual participants in auctions conducted in the marketplace. These agents are created by the C-agent inheriting the initial knowledge concerning their user's preferences, i.e. the level of trust , the maximum price they are allowed to spend for an item, the number of items they should intend to acquire etc. They react to stimuli by both the A-agent, informing them about the progress of the auction they participate in, and the C-agent, instructing them to continue bidding or postpone their execution in case this serves best the session's evolution (for example when another auction shows better prospects or in case of shilling behaviour). CL-agents are not specialized according to the type of the auction that has been assigned to them (English CL-agent, Vickrey CL-agent) the only difference between those two cases is in the auction's behavior for contacting the auction all the other characteristics and behaviours of the agent stay the same. The CL-agent follows the bidding strategy that the trust level decided he must follow but the strategy is also affected by parameters like time and remaining auctions (aggressive, passive, greedy, last-minute bidding etc.). They also have two roles they either just watch the auction's progress and inform the C-agent or take place in the auction and inform the C-agent at the same time. They all possess the same initial characteristics and knowledge with their corresponding C-agent, but present differentiations in their behavior, which explains the reason for their characterization as *clones* of their C-agent.

Graphical Agents(G-Agent)

The G-agents are the agents that remain at the place where the human Client started creating his agents. This agent is responsible to keep communicating with the user interface and updating it according to the agent's (C-agent or A-agent) working progress. This agent was created because all other agents leave the place they were created and move to other destinations after their initiation and this resulted in the loss of communication with the user interface so the G-agent came to solve this problem. The G-agent also creates and updates the progress charts dynamically.

As it has already been described, the multi-agent layer of the platform's infrastructure is devoted in the realization of the negotiating part of the system. The platform combines the elements of agents and agent-rooms to structure a well-organized morphology for the entities that comprise this layer. The design of this structure, along with the fundamental interactions between the various components, is presented in figure 14. Agents are instantiated and interoperate in the rooms.

For each new auction that an auctioneer initiates, a new room/container is created at the pc where the human user sites, named after her/his identifier followed by the code name of the session and ending with the word "temp". This place hosts the G-agent permanently and the A-agent until he is fully initialized. After the A-agent is fully initialized he is moved to another room which is named after her/his identifier followed by the code name of the session and begins with the prefix "Room-for" that is where the agent is going to preside the auction and

all the CL-agents that will register in it. The room of the A-agent is going to be in a host that we assume will always be available.

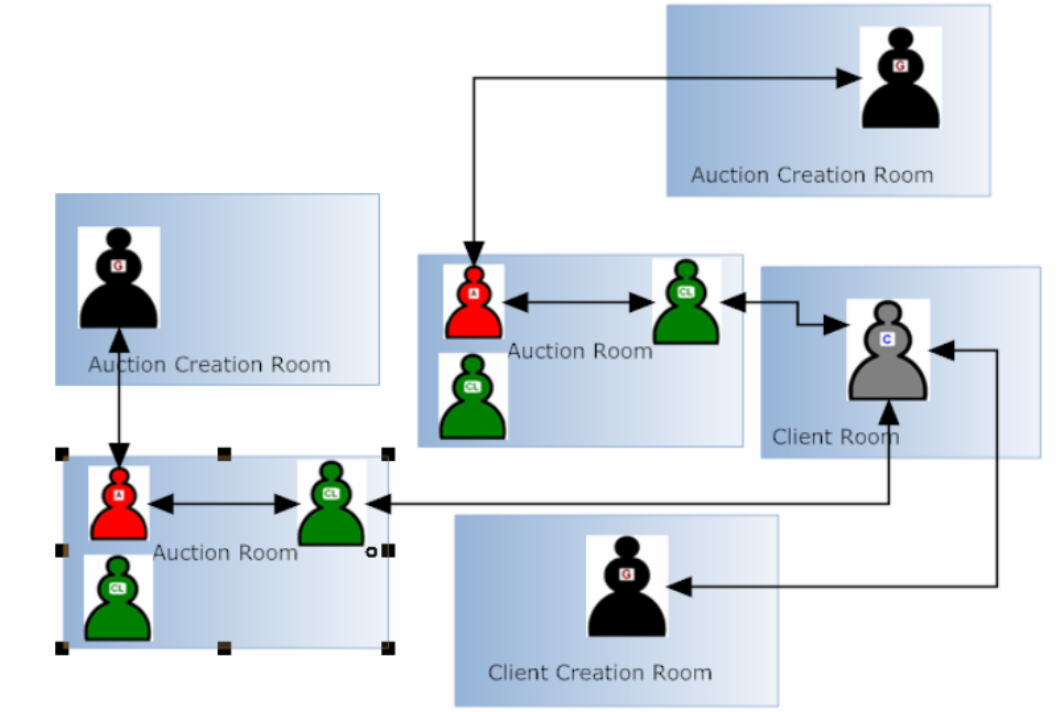


Figure 14 Types of Agents and their communication

Respectively, when a customer starts a new session, a room is created having as name the identifier of the user followed by the name of the session and ending with the word “temp”. In this place the G-agent permanently and the C-agent until he is fully initialized and the first trust level is complete. After that the C-agent is moved to another room which is named after her/his identifier followed by the code name of the session and begins with the prefix “Room-for” that is where the agent generates the appropriate CL-agents and routes them to the places, where the auctions are actually conducted. This technique grants flexibility and efficiency to the structure, because it takes advantage of local interactions between agents during the process of negotiation in an auction.

The following example illustrates the five basic steps of communication among the platform’s agents and the actions that are involved (see also figure 14).

1st Step: An auctioneer decides to create a new auction. The platform creates a room where it creates the G-agent and an A-agent that operates under its owner’s preferences (i.e., the auction type that the user desires, the starting and ending time of the auction, the reserve price, the increment bid step, etc.).

2nd Step: Another room is created where the A-agent is moved and where the auction for the specific artifact will be contacted

3nd Step: A customer decides to find some auctions to participate in. The platform creates a room where it creates the G-agent and a C-agent A the domain the client is interested in is given and the results appear. The customer selects two auctions in the network that s/he is willing to participate in and then he gives the matching criteria he wants the items to have. The Client gathers information about the auctioneer and decides on a level of trust.

4th Step: The platform creates a new room for the c-Agent where he is moved. The C-agent creates two CL-agents, who get from the C-agent information about the auction that they will be appointed to and the bidding strategy decided. The CL-agents possess the knowledge concerning their owner's preferences. After their initialization, the C-agent routes the clones to the auction place that corresponds to each one of them.

5th Step: The CL-agents register in the auction and negotiate locally with the other participants and the A-agent, informing their C-agent about their progress. The C-agent checks for shilling behavior.

4.2.3 Peer-to-Peer Character

The peer-to-peer layer is the cornerstone of our platform, upon which all its functionality is structured. Its peer-to-peer network is a computing resource sharing, knowledge sharing and asynchronous message passing system that implements the virtual auction marketplace environment. The network's topology promotes scalability, while its flexible design makes it a useful device for researchers to experiment with other trading scenarios, as well (for example Web Service discovery and composition).

Two are the basic types of peers: customer and auctioneer peers.

As said, in 4.1.1, the DF (Directory Facilitator) is a mandatory component of an agent platform that provides a yellow pages directory service to agents. Every agent that wishes to publicize its services to other agents, should find an appropriate DF and request the registration of its agent description. There is no intended future commitment or obligation on the part of the registering agent implied in the act of registering. For example, an agent can refuse a request for a service that is advertised through a DF.

Additionally, the DF cannot guarantee the validity or accuracy of the information that has been registered with it, neither can it control the life cycle of any agent. An object description must be supplied containing values for all of the mandatory parameters of the description. It may also supply optional and private parameters, containing non- FIPA standardized information that an agent developer might want included in the directory. Due to the fact, that one can implement their own DF agent, JADE offers the freedom to make peer selection

as complex as one wishes. Thus like in JXTA, there is a standard DF delivered, however one can choose to use another.

Customer Peers

Customer peers serve as single end users, allowing participants to obtain an identity on the platform and gain access to its knowledge space. Every node on the network, including customer peers, contributes to increasing the level of connectivity to the overall network, due to its ability to cache locally DF advertisements of published resources and automatically deliver them to interested peers upon request without any need for human involvement.

Auctioneer Peers

Auctioneer peers are equivalent to customer peers in matters of platform status, but present different functionality. They provide auction services to the platform and share their underlying ontologies for common use. Therefore, they are always accompanied by an ontology database, describing the selling products. They publish the domain and the address by subscribing to the DF.

One deference of our work from SeMPHonia in which ours is based is that the peer to peer Character of the platform is not managed seperately but is managed from JADE too.

All the human user has to do when joining the network is give the IP of the Main Container of the JADE platform and the ip of the place where he wants the agent to move after its initiliazation has finished. The peer groups that where created in SEMPHONIA are now replaced by the agent rooms we described in the previous section.

4.2.4 The platform

All the previously described entities, components and features are integrated in the platform to implement a complete and well-defined e-trading environment. This section presents the overall system infrastructure.

The following example illustrates a typical transaction in the platform environment. Let’s assume that a retailer desires to auction a copy of Da Vinci’s Mona Liza. The first action would be to create one ontology for capturing all the necessary information concerning the specification of the painting (for example, size, technique, theme, etc) and describing the auction template (for example,

auction type, start and end time etc.). This ontology should be stored on a database. Once the preparatory work is completed, the user initiates the Auctioneer Application for registering with the network. A new room/container is created and connected with the platform to obtain global access. In order for the system to generate the new auction session, the only information required by the human user is the IP address of the Main Container and the IP address where he wants the auctions room to be created; all the rest are performed automatically. These include the following actions: first of all, the G-agent is created then the A-Agent is created and he sends a DF advertisement to the DF agent where he describes the ontology of the products that the auctioneer supplies (Ontology Advertisement). This advertisement is characterized by the domain of the product (artifact) and holds as only information the URL of the ontology in order for customer peers to contact the ontology and get the auctions and the artifacts details. Then another room is created at the second IP the user gave and the A-agent is moved there as that is the place where the auction will actually take place. So now the A-agent is ready waiting for participants to take place in his auction.

At some other place on the network, a customer is interested in purchasing a copy of Da Vinci's Mona Liza. In order for him to register with the network, he initiates the Customer Application and he gives the IP of the main container and the IP of the place where he wants his C-agent to be moved and automatically a new room is created locally that is connected with the platform and where the G-agent and the C-agent exist. The customer's first action would be to discover ontologies describing 'copy of Da Vinci's Mona Liza' auctions and, among the available, search and locate the auctions that interest him most. The first task is accomplished by searching the DF for Ontology Advertisement, while the second by searching among the available results that appear in the user interface. In order to initiate the search, the only information needed is the domain of the products he is interested in. The C-Agent constructs and sends to the DF a DFAgentDescription which contains information about the domain he is interested in and so he seeks for advertisements that refer to that specific domain. All responses received contain information about the Ontologies of the agents that the customer should contact in order to access and get the auction's data. Using the Semantic Engine graphical user interface, he is free to view the details to all the ontologies he wants to and select the ones he wants. The information and details are presented to him with the use of OWL-API.

Once the customer concludes on which auctions he prefers, the next step is to see how trusted the auctioneers are. So he contacts all the agents he trusts and gets information about those auctioneer and also the user gives his preferences about how the artifacts wants to be(size, technique etc). From these information with the use of JESS the agent decides how trusted or not the auctions are and how much they match his criteria. After this step the C-agent moves to a new room that is created for him in the second IP the user gave. The final step would be to contact the auctions running for these paintings, join them and start a new bidding session. These tasks are performed automatically by the

application, requiring as sole input the user's preferences, concerning that session (i.e., the maximum price his is willing to pay etc.). The C-agent created CL-Agents for each of the selected auction and gives it information about the level of trust the max price given and how aggressive or not the clone must be. Then the C-agent sends the clones to the remote rooms, where the auctions are conducting. At frequent time intervals, the agents inform the user about the progress of their auctions through the user interface and the G-agent.

From that point on, communication is conducted by the agents. The A-agent informs all CL-agents registered in its auction about new bids, which in turn inform their C-agent. The last tries to find shilling behaviors and decides on the strategy that should be followed during the session, when multiple auctions are involved, and informs the customer's application and also interacts with the G-agent in order to keep the user interface updated. The customer is free to terminate this session at any time, instructing his agents to stop placing bids. He is also given the ability to initiate new bidding sessions with the same or other auctions.

At any given time, peers can leave the network without causing any damage to their sessions. The agents that have been assigned the task of representing their owner's interests on the negotiations exist on rooms that operate on remote hosts, which are presumed to be available at most times. Even when a peer is not connected with the network, its running agents continue to live and interact at those remote rooms. Peers can reconnect at any time in the future to update their knowledge about the progress of their sessions. In addition, all the published DF advertisements remain present in the network, independent of their agent's connection status.

5. Other important features

Our platform is a multi-purpose virtual market architecture that includes numerous advanced features for supporting human users in accomplishing electronic negotiation tasks. This section discusses some additional facilities that the platform integrates, which provide the supplementary infrastructure needed to become a complete and innovative system.

5.1 Flexible Agent Design

In chapter 4 we have described the design of the platform's agent architecture for multiple auction participation, according to which one agent (C-agent) is responsible for managing the session, while a number of other agents (CL-agents) register in the auctions and place bids. It has been made clear that the client's negotiation strategy in each individual auction is expressed exclusively by the CL-agents and their inference mechanism. Therefore, a CL-agent is invoked and has different behavior according to the type of the auction (English CL-agent, Vickrey CL-agent) the trust level and the bidding strategy that depends on the remaining time and the number of remaining auctions (aggressive, passive, greedy, last-minute bidding etc.). The C-agent's role is to pursue the purchase of the desired product in a manner consistent with the client's preferences and agreeing a good deal among the auctions.

This design is based on the design of SeMPHoNIA and can provide high levels of flexibility and scalability to the platform. The C-agent has the ability to custom CL-agents; any agent is offered the ability to employ effective and novel strategies that match its strategy and method. Custom-made CL-agents can be used by agent researchers to experiment with new AI tactics and bidding algorithms without having to re-design the whole negotiation scene; the modification of the agent's reasoning function is adequate. The only compromise that must be made is to preserve the interoperability between CL-agents and the other components of the platform network (user application, C-agent, A-agents). For users with no programming background on the other hand, CL-agents could be handled as black boxes, characterized by their codified features (i.e., protocol support, implemented strategy), and exchanged between other users as separate packages to enhance their application's capabilities.

The important thing to note is that the extension of the platform with new negotiation protocols, even when they present such vast differences in their operation requires no modification in the platform as a whole or in any of its elements. Only the creation of a special Agent Behaviour that will be added to the CL-Agent for this auction is needed. Neither the C-agent nor the A-agent needs to have the protocol hard-coded explicitly beforehand.

5.2 Setting level of trust

After the client selects the auctions he wants to participate in follows the phase of trust management. The first two actions the C-agent does in parallel is create JavaBeans from the OWL ontologies and Contact the Group-Agents he trusts in order to get information about the auctioneers he is going to interact with.

The group agents give him the following information about the auctioneer

➔Trust
➔voters
➔ratio
➔avgSalesPrice

- Trust is the level of trust of this agent and can be from level 1 which is the lowest to level 5 which is the higher. If we don't know the level of trust it is assumed to be level 3
- Voters is the number of users who have given feedback about this auctioneer
- Ration is the grade the auctioneer has gotten from the users and can be from 0 to 10
- avgSalesPrice is the average sale price of the items the auctioneer has sold

If the A-agent trusts more than one group the average value of all the answers is calculated.

We also take for granted that there exists a Price-agent that gives to the C-agent the average price items are sold.

After that information is gathered they are added to the JavaBeans the C-agent has created for each one of the auctions.

But the user must not always blindly accept the facts the groups provide him. So this is where rules come to check these facts and how valid they are.

There are two different types of rules in this level

- Rules about the Ratio and the Number of Voters of the auctioneer

- Rules about the AvgSalingPrice

Rules are implemented using JESS and described in paragraph 7.3

5.3 Finding Shilling Behaviour

Shills, or "potted plants", are sometimes employed in auctions. Driving prices up with phony bids, they seek to provoke a bidding war among other participants. Often they are told by the seller precisely how high to bid, as the seller actually pays the price (to himself, of course) if the item does not sell, losing only the auction fees. Shilling has a substantially higher rate of occurrence in online auctions, where any user with multiple accounts (and IP addresses) can shill without aid of participants. Many online auction sites employ sophisticated (and usually secret) methods to detect collusion.

In our Platform as we have described a client can take part in many auctions he chooses to monitor together and win only one. As we explained he takes active role in one auction at the time starting from the one that finishes earlier and finishing with the one that finishes later.

A client could be accused of shilling only if he monitors two or more auctions that sell the exact same items (e.g. two copies of a painting that are exactly the same and sold in different auctions) and insists on bidding in the auction that has the highest price. Because if a client monitors auctions that offer different items (e.g. two copies of a painting created from different artist and with different size) and he insists on bidding in one of them only he cannot be accused of shilling because his preferences may match more with the one he bids in.

In order for our approach to be able to detect shill we made the following assumption. If the client monitors auctions with the exact same items and the price of the active auction exceeds the price of another auction with the exact same item the CL-agent of the active auctions goes to monitoring mode until the active auctions price becomes lower than the others again.

We needed to take under consideration that the C-agent is responsible to find shilling agents as he is the one that has information from all the CL-agents. Each CL-agent is only in contact with his creator C-agent and the A-agent of the auction he is responsible about. So the user must know that from the moment the C-agent receives a bid information that will fire the rule that makes him want to take the active CL-agent from active to monitor mode until the moment he informs actually his CL-agent to go to monitor mode the CL-agent may have placed other bids too. This number of bids is one and this bid will be thought from the C-agent as shilling bid so an agent may be wrongly accused to be a shiller. In order to face this problem each agent taking part in an auction has the right to do up to two shilling bids in the specific auction. After he exceeds this number he is officially thought to be a shiller.

Another problem we faced when detecting a shilling agent and wanting to stop taking part in an auction was that there is always the possibility that if the CL-agent responsible for this auction is active and just monitoring he may be winning the auction. So the problem would be that if the C-agent stop participating in that auction after finding the shiller and started taking part in the remaining auctions he might win in more than one auctions (the one that he found the shilling and another one). In order to solve this problem the C-agent instead of making his CL-agent abandon an auction the moment he finds a shiller he just sets the CL-agents max price offered to 0 and so it is sure that either he is winning or not in the next round of bids he is going to refuse to offer more and so he will stop taking part in the auction.

If the CL-agent of the auction where the shiller was found is just monitoring an auction. The A-agent is removed from the C-agent's list and the CL-agent responsible for this auction starts taking part in the auction and is deleted.

5.4 Synchronization Policies

For systems, such as ours, that instantiate open distributed environments or manage many simultaneous auctions, synchronization between peers is a critical issue. The nature of such applications necessitates a design that tolerates network and system disruptions and realizes cross-platform accuracy utilizing careful timekeeping procedures. The policies we followed to ensure to ensure synchronization and accuracy at a satisfactory level are the ones that were followed on SeMPHoNIA. This section explains them briefly.

JADE which was used in our platform is an asynchronous environment; therefore a global reference time is a compulsory requirement to keep track of the flow of messages exchanged between remote entities. So all agents refer to Greenwich Mean Time (GMT), while all messages are timestamped to ensure the platform's fidelity. Thus, auctioneers are able to schedule events and determine the validity of message sequence occurring during them.

All applications invoke the appropriate Java objects to implement local timekeeping. Time reflection using Java classes, though, depends on the host environment of the Java Virtual Machine [47]. This approach introduces an important side-effect; peers across the network are not synchronized and even agents created by the same user, but running on different machines, maintain different time images for the same snapshot. A synchronization algorithm has been developed for that purpose that is invoked during the phase of agent creation and aims at determining the time interval between the agent's runtime environment and the host system that initiated the agent. The algorithm is based

on the Ping/Pong procedure, according to which the agent sends a ping message to the user's platform and measures the time needing for the response to arrive, estimating network delays and the declination between the two clocks.

Besides that, even the design of the agent negotiation scheme supports accuracy during auction execution. The ability of CL-agents to migrate to the place, where the actual auction is conducted, provides them with the flexibility to exploit local interactions. All unpredictable message delays due to network traffic are avoided, but, most important, all agents refer to the same system clock, since they operate on the same room. In addition to this, they are able to apply accurate algorithms for dynamically calculating the length of their bidding determination cycle and improving their strategy. To do so, they take into account variables and parameters of their local host that are considered to be more stable, because they always depend on the same computing resource.

5.5 Autonomous execution

Great emphasis has been given in generating user-independent sessions that automate most of human procedures, even the ones that require a certain degree of intelligence. The platform has been designed to support human interference in sophisticated tasks and to preserve user-created executions in an autonomous manner, even when the user is not connected on the network.

Automation is accomplished by exploiting certain features of agent technology. Intelligent agents are able to inference, deliberate and communicate without the need of human guidance but rather based on a set of start-up parameters, on changes occurring in their world and on a set of inborn knowledge or beliefs [48]. Such agents are utilized here to fully automate the user's participation in multiple auction creation, monitoring and bidding. Users (either customers or auctioneers) are free to go offline, after initiating a new session, and trust the execution on the "hands" of software agents. The platform offers a mechanism for connecting and disconnecting with the system, maintaining profile information locally for all users, so that they can keep track of their sessions' progress and recover execution whenever they desire to.

This chapter presented a number of additional functionalities of our platform. Their intention is to provide the means for assisting users in accomplishing electronic trading task and enhancing their interaction with advanced techniques. The next chapter describes the steps for the creation and

execution of auction services in SeMPHoNIA, providing extensive walkthrough examples and screenshots.

6 Creation of the auction services

This chapter contains directions for participating in the platform's environment. It outlines the steps users must take and the information needed to play the role of the operator, auctioneer or customer in the system. Extensive guidelines and screenshots are presented, concerning all available functions offered to the user. The screenshots are taken from the Windows XP Professional platform, therefore minor differences, regarding the window layout might be present, compared to other systems (i.e. UNIX).

6.1 Requirements

In order for the platform to work there are three main requirements

- The Main container of the JADE platform must be created in a specific host who's IP the users know in order to connect with it.
- Some trusted agents are needed that will give the information needed in the trust management level.
- We also need to suppose that we know the IPs of some hosts that always remain active. They will be used as the hosts well the room that the A-agents and C-agents move after they are initiated.

6.1.1 Setting the Main Container

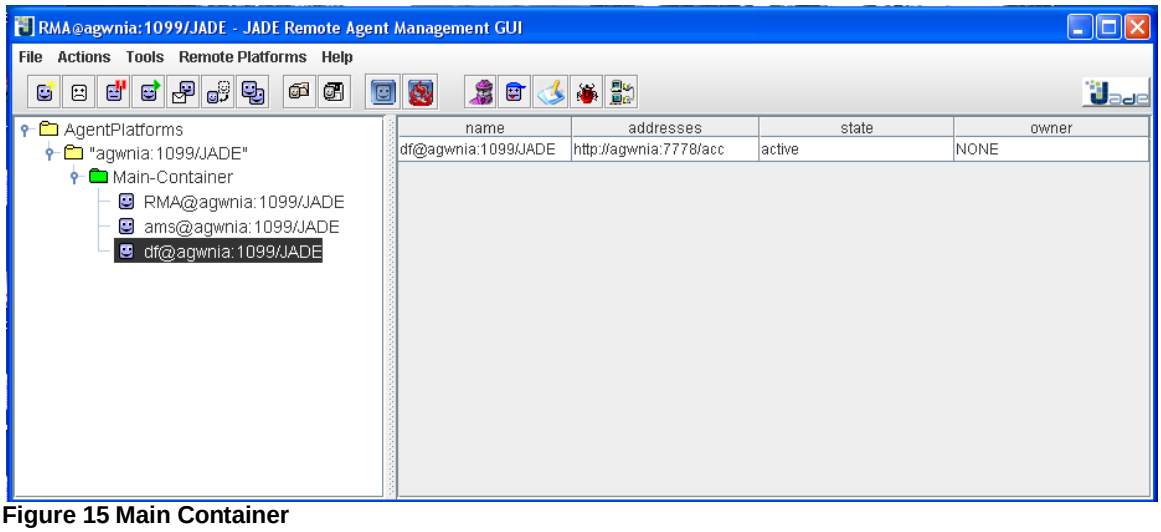


Figure 15 Main Container

The jade platform is initiated in a specific host that the users will know and use to connect to the platform. At this host with the initiation of the platform as we can see in figure 15 two agents are created the AMS (Agent Management System) agent that is responsible for providing 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) agent that provides the Yellow Pages service by means of which an agent can find other agents providing the services he requires in order to achieve his goals.

6.1.2 Group Agents

We also create agents that are responsibly to give information to C-agents about the reputation of auctioneers their activity up until know and the level of trust they have according to other users.

So it is also assumed that such agents exist in the network all the time and provide the information given but also get feedback from the C-agents that they trust.

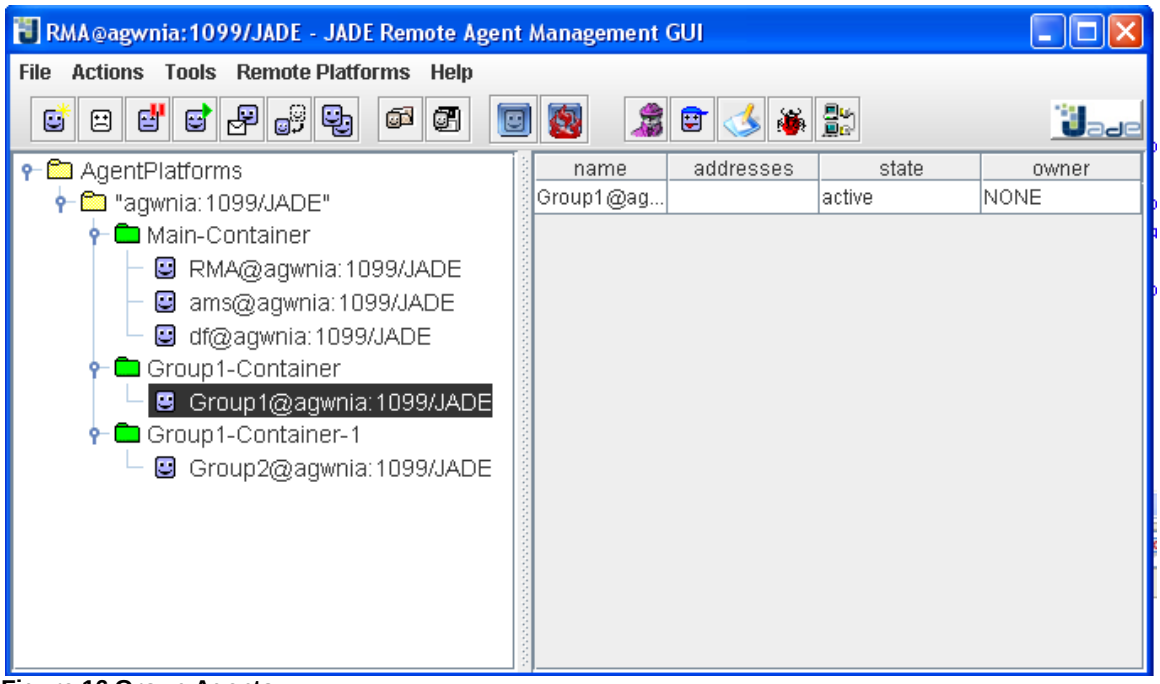


Figure 16 Group Agents

6.2 Auctioneer Application

In our platform the same facilities and privileges are offered both to auctioneers and customers, therefore the interaction with the system and the mechanisms triggered by human users follow similar patterns. The auctioneer application connects to the JADE platform, creates a new room in the users host and inside the room one can find the G-agent. The application is able to publish one or more OWL ontologies, each containing metadata about one or more products. In addition, supports users in initiating, monitoring and terminating auction sessions. For each ontology published a different A-agent is created. The A-agent that is responsible for shriveling and protecting the auction execution on behave of the human user is capable of handling two types of auction; English and Vickrey (please refer to section 4.2.2 for information about A-agents and the corresponding auction types).

The first time the auctioneer application is launched the user is requested to specify the communication IP address of the main container and the IP of the host where he wants to create the auction's room. He also gives the URL of the ontology he wants to publish.

The next time the user logins to monitor his auctions he want need to give the IP of the main host as it is kept in a file along with the A-agents AIDs he has created in order for the G-Agent to regain communication with them.

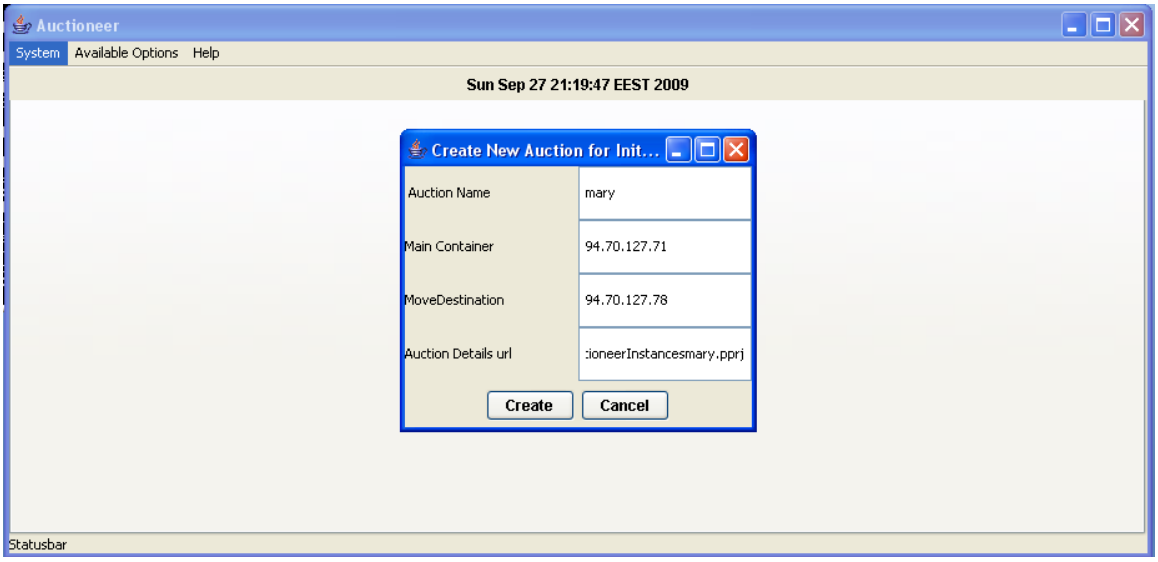


Figure 17 Auctioneer Login

Before any person can interact with the platform, s/he must first be appropriately registered, by providing a unique identifier through the process of login (figure 7). This identifier is kept by the DF agent at the main container. After the user logs in the environment, if the file with his past actions is not empty the system seeks to regain communication with the agents he had left active. The

application knows where to look for, because it keeps information about A-agents AIDs that a user had created in the past, which show all of her/his remote auctions. Then, it informs the user about the result and triggers the A-agents' "reconnection" function, in order to inform the application about the auctions' current status. Respectively, the user may choose to either logoff or exit at any time, leaving all of her/his sessions running at the remote rooms.

The process of creating an auction is fully automated and human users are completely relieved of unnecessary and complicate procedures. Indeed, initiating a new auction is accomplished in one step, provided that the product's OWL ontology has already been created and we have its URL. To construct a new auction service, users are just prompted to indicate the URL of the ontology (figure 18). On receiving a new ontology, the system initiates a set of actions that involves:

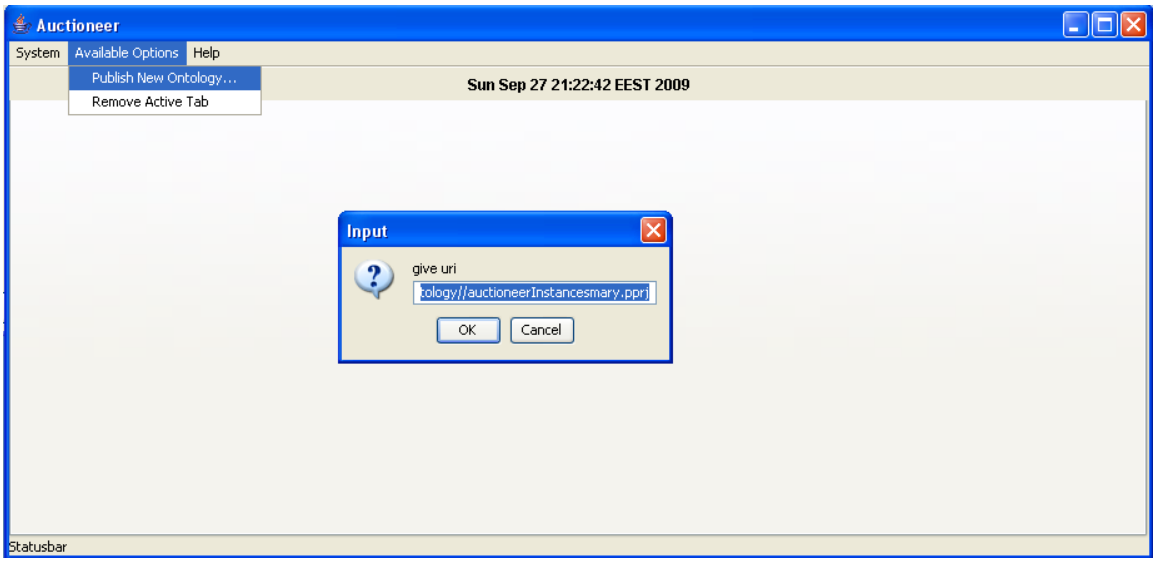


Figure 18 Auction Creation Step 1

The application takes the URL and with the use of OWL API extracts all the necessary information for structuring the new auction, such as the resources of the items that are for sale, the type of auction (English, Vickrey), the start and end times of the session, the increment bid step, the reserve price etc. The system presents the results to the user and waits confirmation before proceeding to the following steps (figure 19).

Then, the application constructs the room where the A-agent will be sent using the remote IP host the user gave him and initiates the A-Agent for this auction with the information it got from the previous step

After its initiation the A-agent automatically moves to its room (figure 20).

When it arrives there the auction starts taking place there and the agent sends to the DF service an advertisement containing its items domain and the auction ontology URL.

The aforementioned steps are repeated for every product the user wants to sell and has OWL ontology with its details.

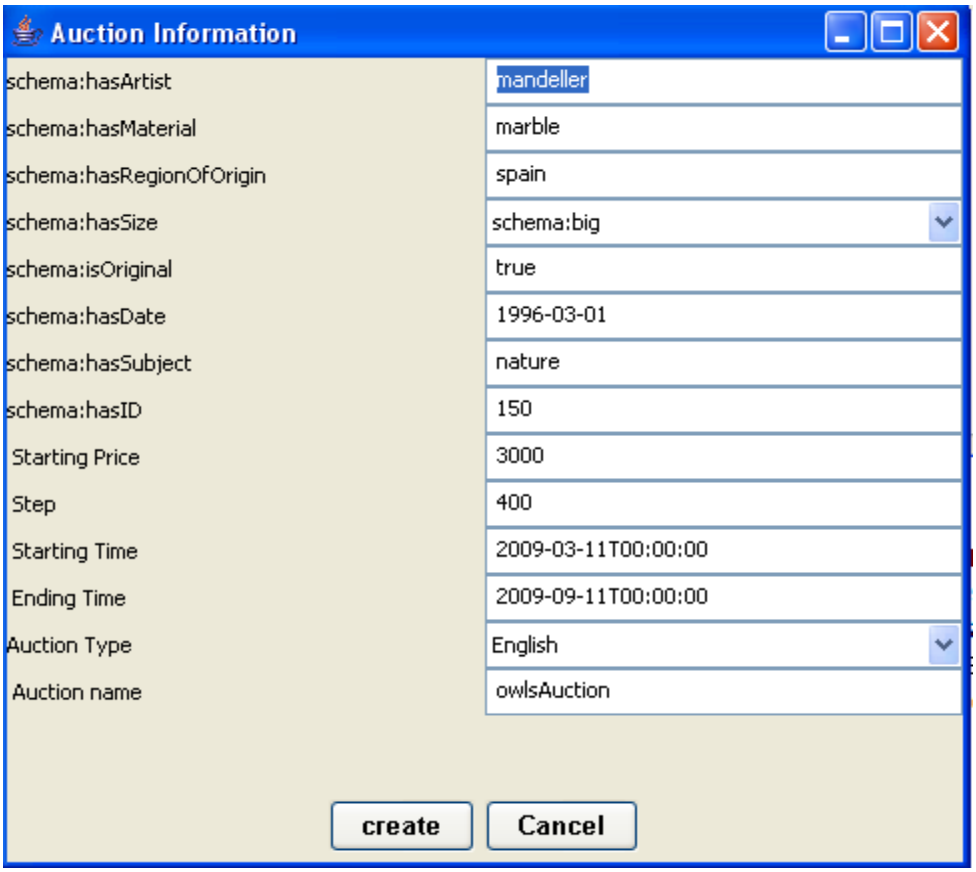


Figure 19 Auction Information

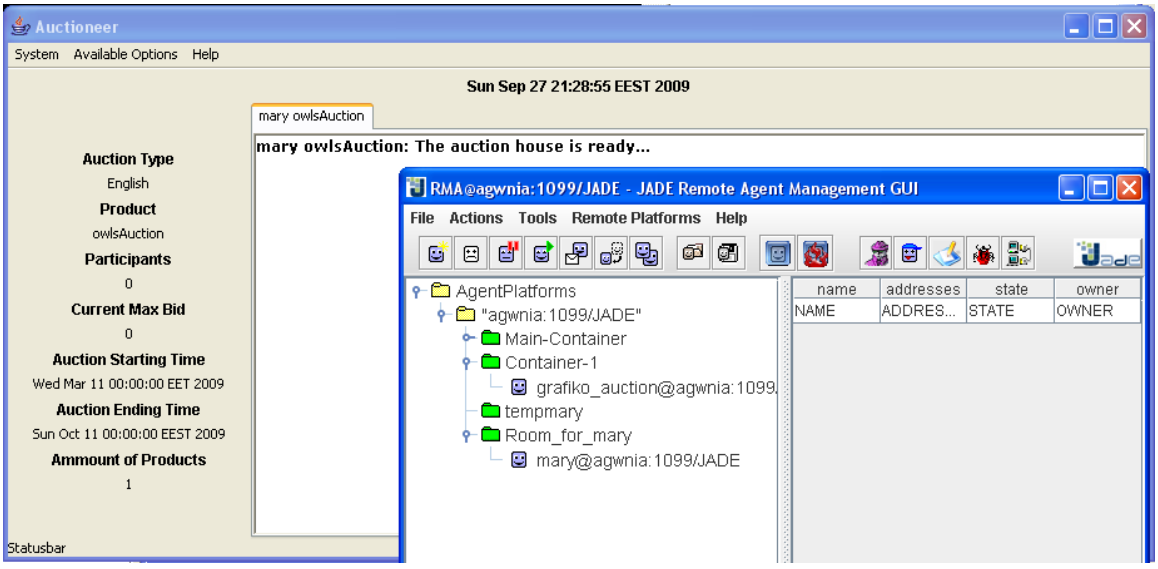


Figure 20 after Auction Initiation

Additionally, users are free to terminate an auction, after it has ended, by selecting the Remove Active Tab choice on the Available Options menu. This will cause the deletion of both the room and the A-agent at the room and, also, the invalidation of the advertisement that was stored in the DF.

6.3 Customer Application

Customers constitute the central point of interest in auction systems and in commercial systems in general, therefore our goal is to make the tedious process of search, trust, participation, monitor and analysis of auctions as simple and complexity-free as possible. The platform offers several combinations of choices to users; they can be active bidders in an auction or passive observers extracting statistical data used in various ways; they are allowed to initiate one or more sessions in parallel and follow their progress; in one session they can participate in one or more auctions, aiming at winning only one of them, when the result of one auction may affect the action taken for the other. The system supports users in following the optimum path, when interrelated auctions are involved.

As with auctioneer applications, the first time the auctioneer application is launched the user is requested to specify the communication IP address of the main container and the IP of the host where he wants to create the auction's room.

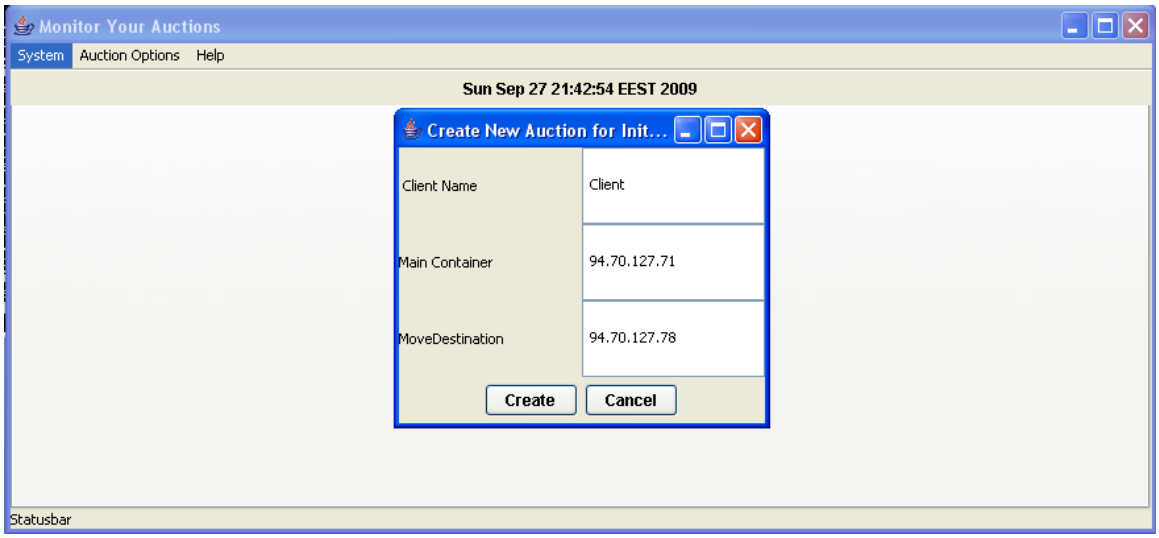


Figure 21 Client Login

The next time the user logins to monitor his auctions he won't need to give the IP of the main host as it is kept in a file along with the C-agents AIDs he has created in order for the G-Agent to regain communication with them.

Before any person can interact with the platform, s/he must first be appropriately registered, by providing a unique identifier through the process of login (figure 21). This identifier is kept by the DF agent at the main container. After the user logs in the environment, if the file with his past actions is not empty the system seeks to regain communication with the agents he had left active. The application knows where to look for, because it keeps information about C-agents AIDs that a user had created in the past, which show all of her/his remote auctions. Then, it informs the user about the result and triggers the C-agents' "reconnection" function, in order to inform the application about the auctions' current status. C-agents preserve information about the state of their CL-agents at all times. Respectively, the user may choose to either logoff or exit at any time, leaving all of her/his sessions running at the remote rooms.

6.3.1 Semantic Analysis

The first and most important step for a customer, when creating a new session, is to search among available products, conclude about their specifications and discover auctions offering the ones that satisfy her/his preferences. We offer a Semantic Analysis Engine and a graphical user interface to assist clients in locating auctions, by getting the ontologies that describe the products offered and appearing their characteristics in the user interface.

On the JADE network, numerous Ontology Advertisements are published to the DF characterized by their domain. Once the user decides on the domain that s/he is interested in, the Semantic Analysis Engine window is displayed (figure 23). At the same time, a DFAgentDescription message is sent to the DF seeking for advertisements that refer to that specific domain. All the responses received are cached locally, containing information about the ontology that the customer can contact, in order to learn the auctions details. A response is received containing all the available auctions at that time.

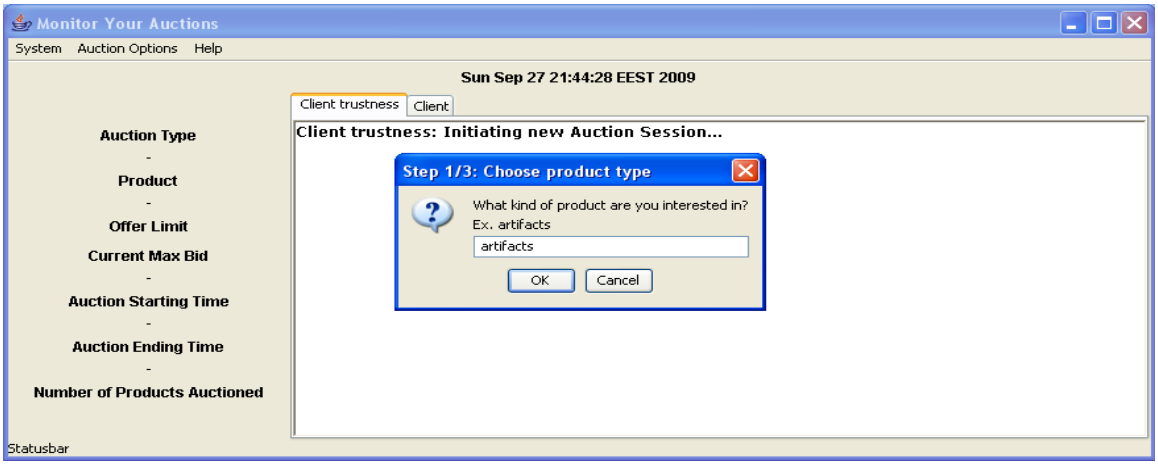


Figure 22 Choosing Domain

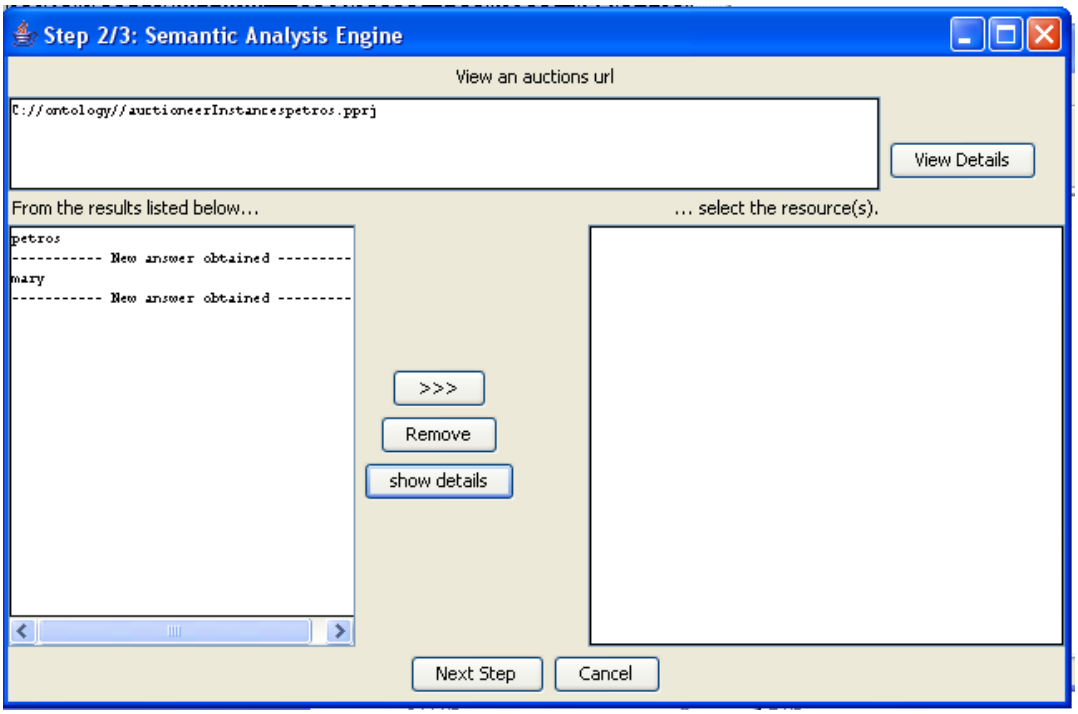


Figure 23 Semantic Analysis Engine

The customer can select a specific auction and view its URL in the text area located at the top of the Semantic Analysis window by pressing the show details button. Then by pressing the view details button he can see the auctions details and if he wants to take part in this auction press add or otherwise press cancel.

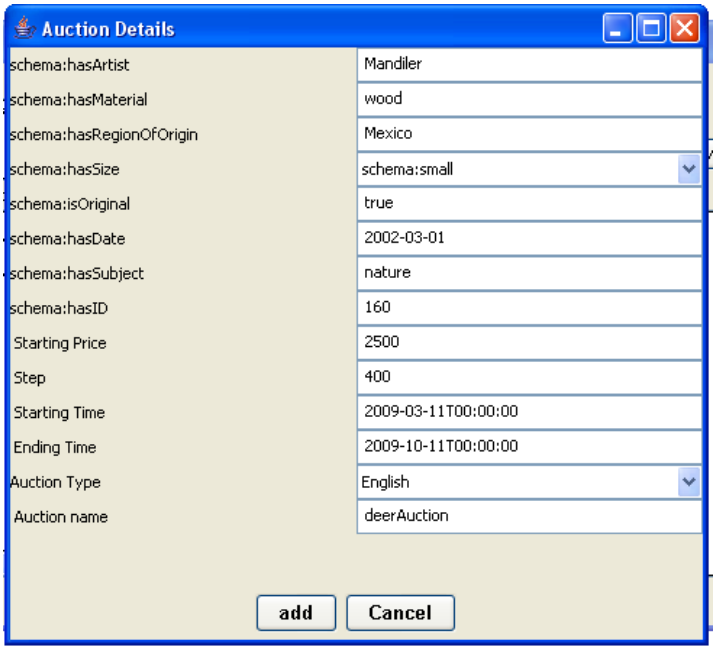


Figure 24 Selected Auction URL's Details

Whenever the user finds a product of interest, s/he should place it in the right text area titled “Products”. The number of resources entered in that box indicates the number of auctions, in which s/he will participate in parallel. If the user is only interested in one product, then only one resource should be at the “Products” box. On the other hand, if the user has located multiple products, but would like to acquire only one of them, then selects the resources and places them in the box. The system will follow the progress of all of them, while the intelligence of its agent design will ensure that at most one of the products will be obtained, scoring the best utility factor possible.

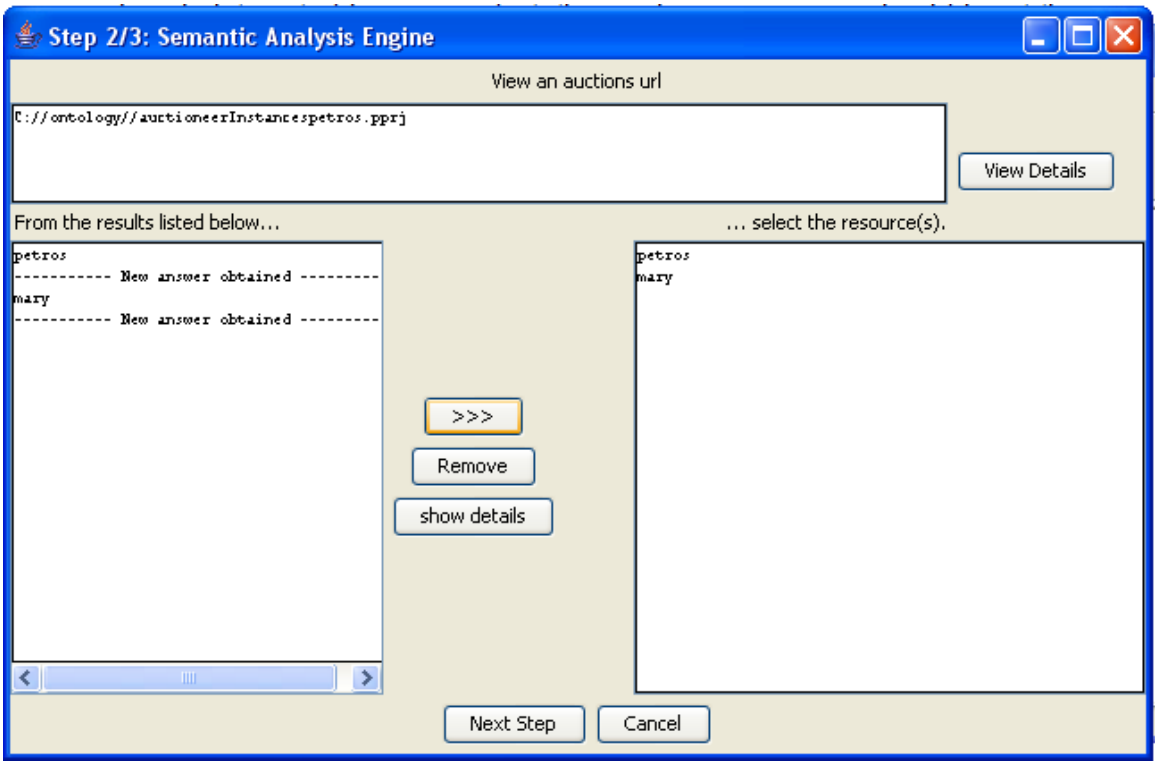


Figure 25 User's selection

6.3.2 Trust management

After the client sellects the auctions he wants to participate in and presses the next button the C-agent communicates with the group-Agents he trusts and get available information about the auctions he wants to partisipate in.

The same time the C-agent with the use of the OWL-API creates the following pop up window and transfers all the ontologies into javabeans that will be used to create JESS facts

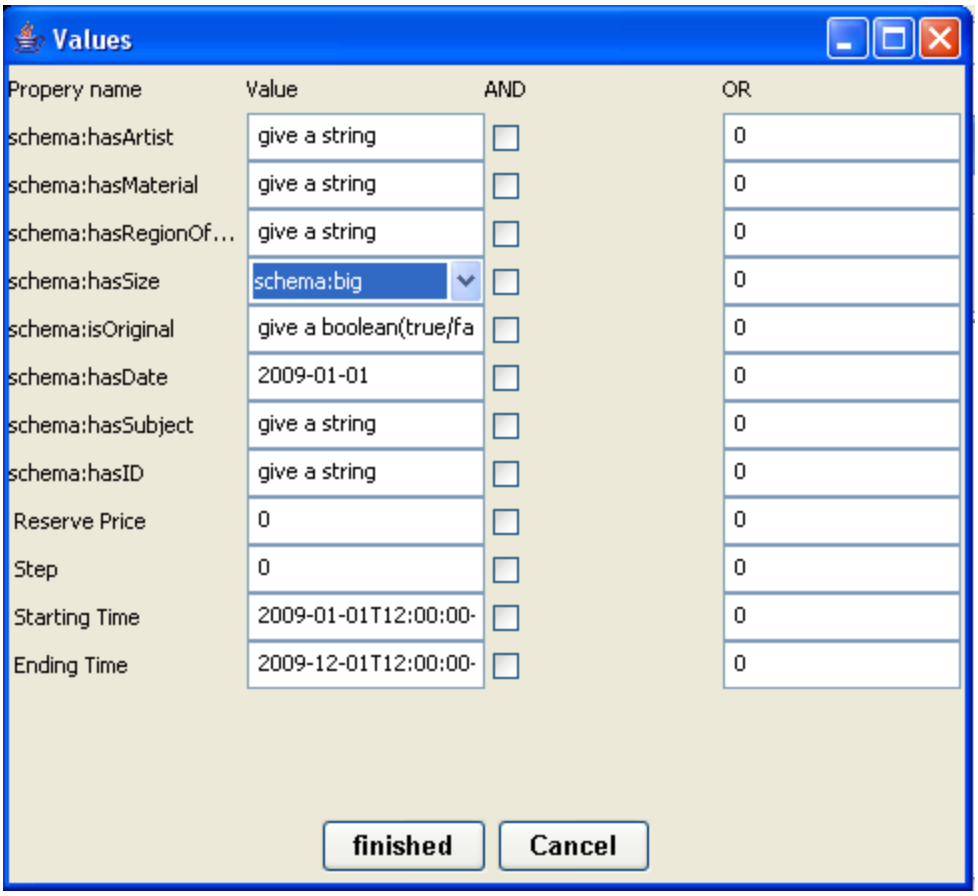


Figure 26 User's preferences

Which contains all the characteristics of the artifact extracted from its ontology and the characteristics of the auction. In the second coloumn This is the window where the client gives his preferences for the artifact he sicks. The characteristics whose tick box is selected are pbligatory to match whereas those who have in their fourth coloumn a value difeerent from 0 are optional but desirable.

After pressing the finished button the preferences dynamic rule is created and along with the trust rules they are applied on the javabeans representing the auctions. The result that comes from that is the trust level for each auction and the max bid that will be applied in each auction.

After this work is done the client can select the auction bidding to begin.

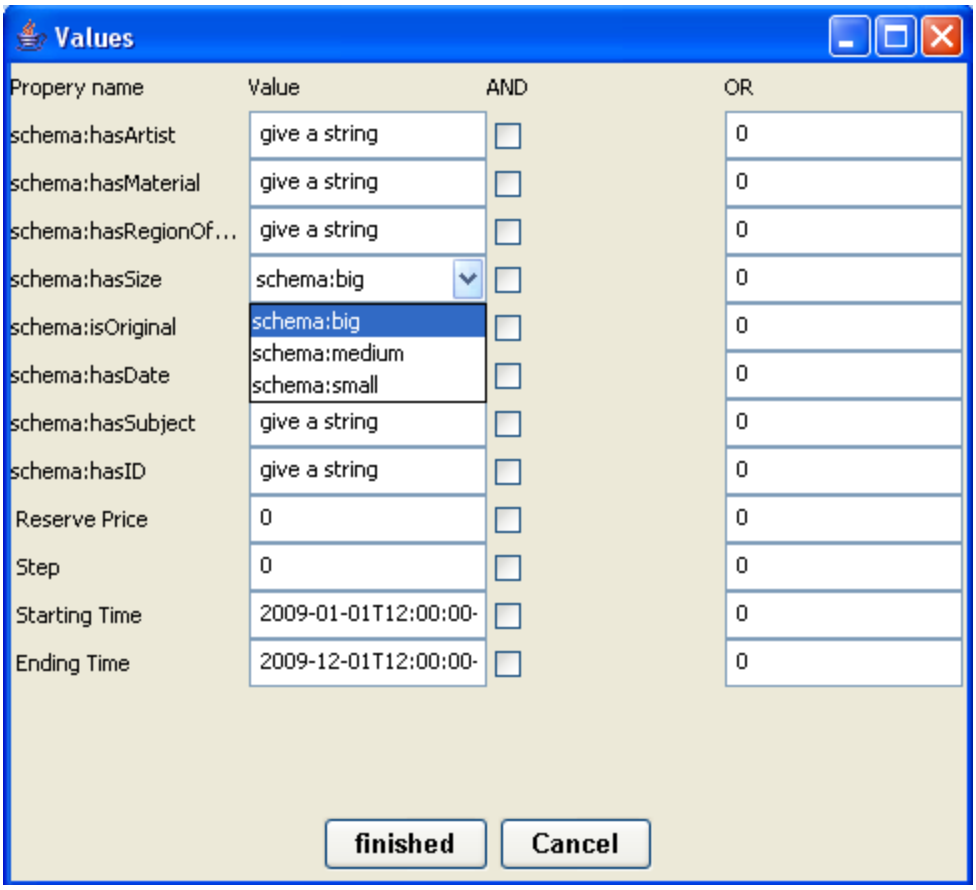


Figure 27 User's Preferences II

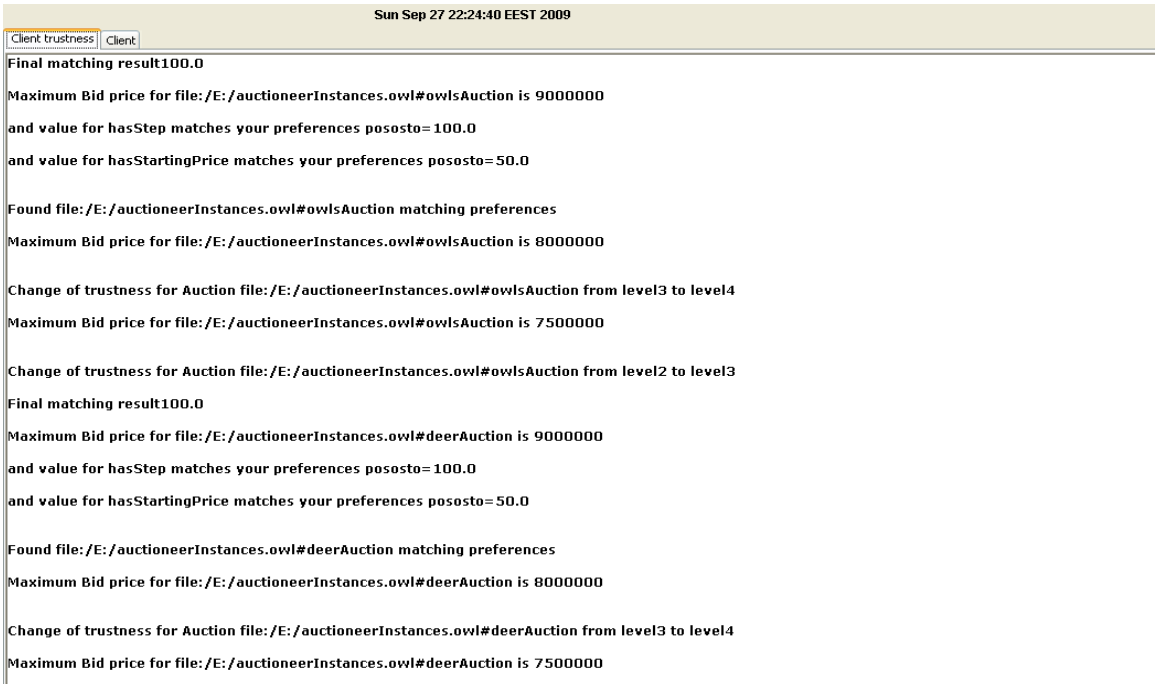


Figure 28 trust rules fired

6.3.3 Bidding in Auctions

The basic functionality that the Customer Application offers to users is to automate the process of biding in one or more auctions. To initiate a bidding session the user selects the “Register in Auction…” choice of the “Auction Options” menu. In each bidding session, the user may participate in one or more auctions and the system will attempt to win one of them, at most. Moreover, the user is free to start as many sessions as s/he desires. The first step in creating a new auction session is to specify the type of products that s/he is interested in (figure 24). This will trigger the system’s analysis mechanism to start searching the network for Ontology Advertisements dedicated to the domain that the user has specified. The next step will be the Semantic analysis described previously followed by the trust management.

After finishing the previous steps, the customer presses the “Monitor Auctions” button and the application activates the following set of actions:

Since the user has nothing else to do it is time for the agent to move so first of all the A-agents room is created at the host who’s IP the user gave.

Then the C-agent moves to the new room this way it is guaranteed that even if the user logs out of the system and closes hic pc the C-Agent will remain active.

After the C-agent has moved, it creates the appropriate CL-agents and routes them to the remote places, where the auctions are conducting. Figure X presents how the agency will look after the creation of the place and the existence of the C- and CL-agents. CL-agents may need to travel to remote agencies, as well.

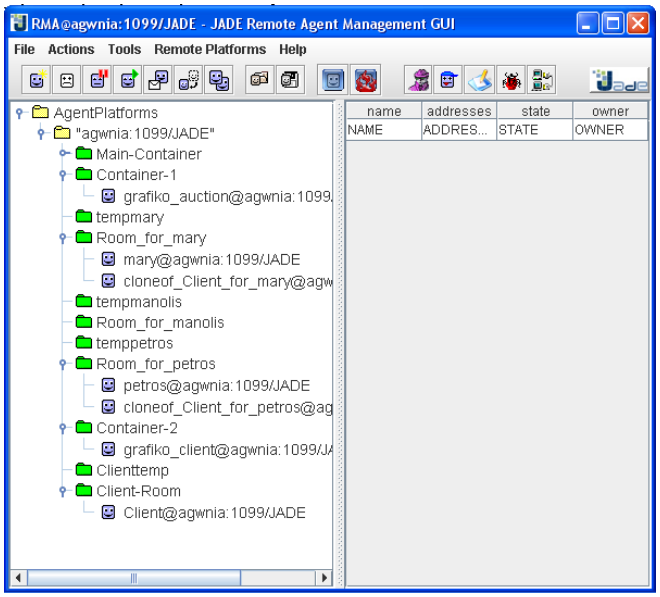


Figure 29 Agency after agents’ creation

Finally, the graphical user interface is updated, with the help of the G-agent that communicates all the time with the C-Agent presenting information about the auctions the user participates in (figure 30,31) and the session as a whole (figure 32). The CL-agents inform the user about the progress of bidding in their auction, while the C-agent's tab displays general messages, such as which CL-agent is currently active, which is holding the maximum offer in its auction, etc. Data are updated in real time, whenever a change occurs and the user is free to browse through all of her/his auctions by clicking the corresponding tab.

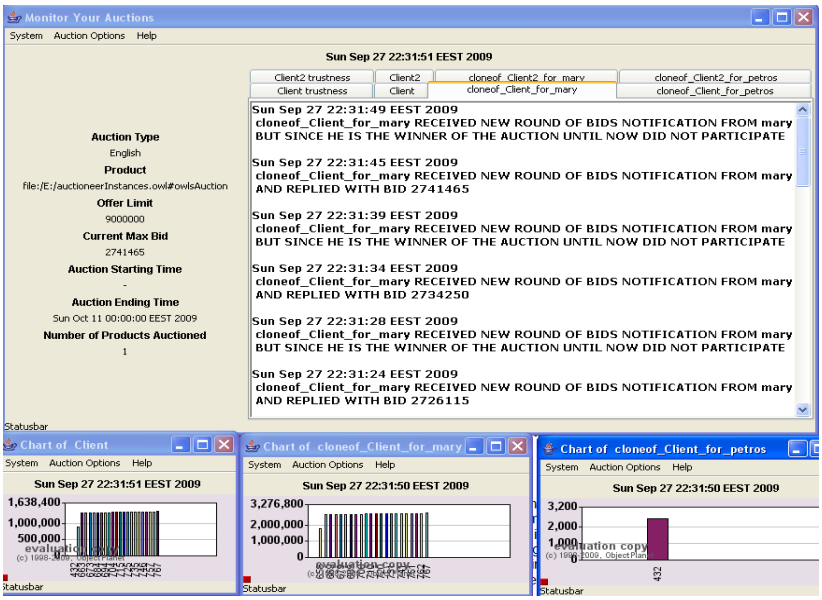


Figure 30 Specific Clone

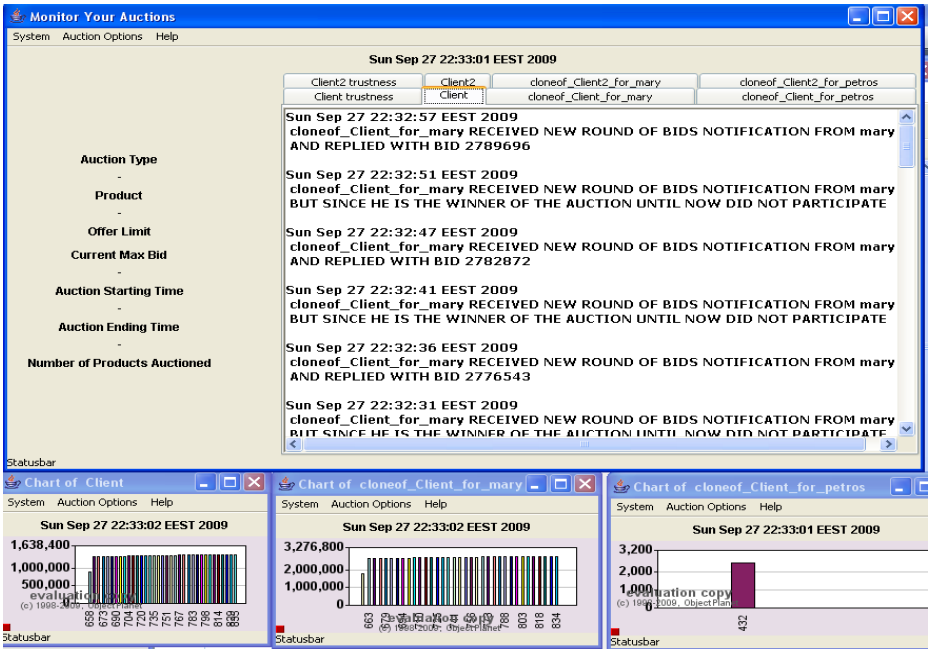


Figure 31 Client

Figure 32 Auction progress

6.3.4 Abandoning Auctions

The user is free to terminate a session, either bidding or monitoring, and abandon the auctions involved. For that purpose, the “Remove Active Auctions tab” choice of the “Auction Options” menu is granted. Terminating a session will cause the corresponding place to be erased and the C-agent along with their clones to be deleted. In addition, the graphical user interface is updated removing the current tab from the main frame. There are certain restrictions that one should consider, though:

To terminate a session, the user must remove the tab of the C -agent. In such a case, a warning message appears, to prevent any unintentional actions.

Removing the tab of a CL-agent will only erase the corresponding tab. The agents continue to exist and operate. This means that a user is only allowed to terminate a session as a whole and not one of its auctions. Users can use this function to lighten their interface from multiple auction tabs. When the user reconnects with the system, all the tabs will re-appear.

Abandoning an auction will cause the user’s registration to be invalidated. However, if at that particular moment s/he was the maximum bidder in the auction, her/his offer continues to exist, until someone else outbids it. If the auction ends with no other offer, the user is considered the winner, despite the fact that the session has been terminated.

This chapter summarized the way users can interact with the platform and explained the basic steps, along with the procedures triggered, for executing auction services. Next, we are going to explain some implementation principles followed in the creation of the system.

7 Implementation

The platform is entirely programmed in Java, in order to minimize dependencies on the underlying operating system and to be consistent with existing software. JDK 1.5 or higher is required for most of its components. The customer and auctioneer applications, as well as, the JADE and JESS components have been implemented on Windows XP platform. All forms and modules used for user interface have been implemented mainly using Java's Swing toolkit and, in less extent, Java's Abstract Windowing Toolkit (AWT) elements. The Ontology and its instances where created using Protégé 3.4.Beta.

7.1 Agent Communication

A set of standard messages that are exchanged between our agents is implemented to describe the minimum required communication needs of the multi-agent layer of the platform. The message invocation is triggered by events occurring in the auction world (auction opening or termination etc.) or by inferences of their own or their master's reasoning mechanism. The description of those messages and their parameters are provided in the context of interfaces that the agents implement in order to ensure cross-platform interoperability, necessary for building a unifying framework. The messages formulate a negotiation core for defining auction activities and are always independent of the particular type of the auction.

For the agent communication through messages we used the FIPA ACL language. A FIPA ACL message contains a set of one or more message elements. Precisely which elements are needed for effective agent communication vary according to the situation; the only element that is mandatory in all ACL messages is the performative, although it is expected that most ACL messages will also contain sender, receiver and content elements.

If an agent does not recognize or is unable to process one or more of the elements or element values, it can reply with the appropriate not-understood message.

Specific implementations are free to include user-defined message elements other than the FIPA ACL message elements specified in *Table 1*. The

semantics of these user defined elements is not defined by FIPA, and FIPA compliance does not require any particular interpretation of these elements. Some elements of the message might be omitted when their value can be deduced by the context of the conversation. However, FIPA does not specify any mechanism to handle such conditions, therefore those implementations that omit some message elements are not guaranteed to interoperate with each other.

The full set of FIPA ACL message elements is shown in *Table 1* without regard to their specific encodings in an implementation. FIPA-approved encodings and element orderings for ACL messages are given in other specifications. Each ACL message representation specification contains precise syntax descriptions for ACL message encodings based on XML, text strings and several other schemes.

Table 2 FIPA ACL message elements

Element	Category of Elements
performative	Type of communicative acts
sender	Participant in communication
receiver	Participant in communication
reply-to	Participant in communication
content	Content of message
language	Description of Content
encoding	Description of Content
ontology	Description of Content
protocol	Control of conversation
conversation-id	Control of conversation
reply-with	Control of conversation
in-reply-to	Control of conversation
reply-by	Control of conversation

For the messages exchanged between agents we followed the directions of the FIPA specifications for these types of auctions (English, Vickrey).

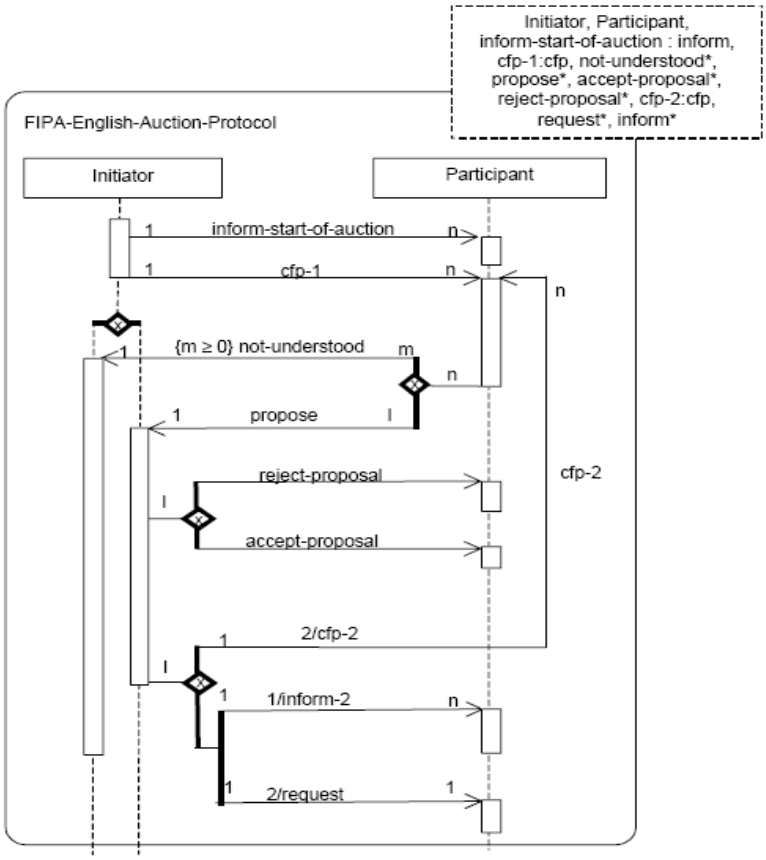


Figure 33 FIPA specification for English Auction

The messages are classified according to the negotiating parties involved as follows:

7.1.1 Auctioneer Generated Messages

A-agent to CL-agents

Messages that the A-agent broadcasts to all participants of an auction to initiate it and manage its execution.

StartOfAuctionNotification(OpeningPrice):

Informs the participants that the auction session has opened with the given opening price.

EndOfAuctionNotification(Winner, WinningBid):

Informs about the termination of an auction, along with the winning agent and the winning offer.

NewBidNotification(MaxBidAgentAID, NewBid):

Whenever a new offer is accepted and verified as valid, the auctioneer informs the rest of the participants. In auction, like Vickrey, that the offers are private, the auctioneer passes the *null* value as the corresponding parameter.

A-agent to G-agents

Messages that the A-agent broadcasts to its G-Agent In order to update its Interface

NewActionNotification(sender, item, message, price, participants)

Each message includes

the localName of the sender which is used to find the right tab in the interface

the name of the Item that is sold

the message describing the action that just happened

the current price of the item

the number of participants in the auction

This type of message is sent to the G-Agent about every action that takes Place in the A-Agent like

A new bid that was accepted

A new CL-agent joining the auction

A CL-agent leaving the auction etc

7.1.2 Customer Generated Messages

C-agent to CL-agents

Messages sent by the C-agent to each clone individually for managing a session and instructing the course of action that they should follow.

Activate():

Activates a monitoring CL-agent to place new offers.

Deactivate():

Deactivates a previously active CL-agent, setting its state to monitoring mode.

Abandon():

Instructs a CL-agent to permanently abandon an auction in case shilling behavior was detected or the C-agent won in another auction.

CL-agent to G-agents

Messages that the A-agent broadcasts to its G-Agent In order to update its Interface

NewActionNotification(sender, message, price, clone)

Each message includes

the localName of the sender which is used to find the Wright tab in the interface

the message describing the action that just happened

the current price of the item

the AID of the clone agent

This type of message is sent to the G-Agent about every action that takes Place in the A-Agent like

A new bid that was accepted

A new CL-agent joining the auction

A CL-agent leaving the auction etc

CL-agent to A-agent

Messages sent by CL-agents to the A-agent for interacting in an auction. The message format is always independent of the type of the particular auction. Clones are characterized by the name of the C-agent that creates them and the name of the A-agent in which they participate.

RegisterInAuction(AgentName, CLidStr):

Message sent by a clone to register as a bidder in an auction session.

The auctioneer will never know if a CL-agent is active or monitoring.

ProposeNewBid(AgentName, newmaxbid):

Placement of a new bid.

RefuseNewBid(AgentName)

Refuse to place a new bid.

CL-agent to C-agent

Messages sent by CL-agents to their master C-agent to inform it about the progress of the auction that they participate in.

ArrivalConfirmation(CLname, CLstate):

Informs the master agent that the migration to the remote room, where the auction is conducted, was successful along with its current state.

InformAboutStartOfAuction(CLname, StartingBid):

Informs the master agent about the initiation of the session that the clone participates in.

InformAboutNewBid(CLname,MaxBidAgentName, MaxBidAgentNameParent, NewBid, auctionItem, message):

A new offer has been accepted in the corresponding auction and the reply the CL-agent gave.

InformAboutEndOfAuction(CLname, Winner, WinningBid):

Informs about the termination of the corresponding auction session.

7.2 Implementation of multi-auction bidding logic

An important component of any intelligent software agent is its inference engine. In our platform as we have described before the client can select a number of auctions whose types may be different (English, vickrey), their starting and ending time may also be different, from all these auctions the Agents must manage to get the best result and at the best price. The bidding process for each auction is managed by its own CL-agent but yet the C-agent continues having communication with all the CL-agents, maintaining knowledge about all of them and has a well-defined dynamic plan about which is the priority of each auction taking also under consideration the results of the trust level, the matching of the auctions characteristics with the clients preferences and the Maximum bid decided for each auction.

Our current approach is indicative and can be used as a reference point for implementing more advanced strategies. The C-agent is designed to manage multiple CL-agents and decide which of them should bid and which should wait, according to the progress of their auction. Four are the basic constraints that the C-agent's logic mechanism is designed to comply with:

→Singularity Constraint. Among all clone agents, only one is allowed to be active at any particular moment and, thus, permitted to place bids in its auction. The rest of them are set at a monitoring mode, just getting the new bids from their auctioneer and passing them to the C-agent.

→Exclusiveness Constraint. While the active clone holds the maximum bid in its auction, no other clone can be set active. Only after the first has been outbidded, another one is eligible to become active. We have also said that even if the user stops a clone, if the clone was winning at that moment the C-agent will wait for another better bid at that auction before taking place in another auction

→Pass from All Constraint. One approach could be to first participate in the auction that's matches more with the users criteria. But this approach has the drawback that the user may lose the chance to participate in other auctions that take place before the auction that has the best match. So in our approach the C-agent takes the auctions from the first to the last chronologically according to their finish time.

→Conservative to aggressive Constraint. The purpose of this constraint is to prompt active agents to adopt a behavior that follows their chances of getting the item from one of the auctions. The C-agent always takes under consideration the remaining auctions before deciding the maximum bid the clone will give to the currently active auction. The active clone always watches the remaining time in order to decide how aggressive he will be and when he will give his maximum offer limit. This way at the first auction the CL-agent are more conservative whereas in the last more aggressive.

These constraints are the core principles of the agent's logic reasoning mechanism that ensure single auction winning and potentially best deal agreement for the customer. They guarantee that no surplus items would be bought or purchases exceeding the user's offer limit would be made.

Figure 34 summarizes the general framework of the algorithm that implements the aforementioned C-agent logic. This algorithm is triggered either by events occurring in the active auction or by a shilling behavior found. No other events affect the progress of the session and, therefore, necessitate no alteration at the state of the CL-agents.

IF a change occurs in the active auction
CHECK the state of the active clone
CASE the active clone has won the auction
THEN *abandon* all other auctions
CASE the active clone does not hold the maximum bid AND (the active clone has reached its offer limit OR the active clone has lost the auction)
THEN among all clones *activate* the next one and turn the currently active one's mode to monitoring
CASE the active clone does not hold the maximum bid
THEN make new bid
CASE the active clone holds the maximum bid
THEN do nothing
IF a shilling behaviour occurs in the active auction
CHECK the state of the active clone
CASE the active clone does not hold the maximum bid
THEN among all clones *activate* the next one and abandon this one
CASE the active clone holds the maximum bid
THEN do nothing until another better bid comes and then abandon

Figure 34 C-agent Logic

It should be noted that many other tactics can be easily created from this approach, by enhancing the reasoning mechanism with new constraints the user may want to use/ Since no optimal solution can be traced due to the variety of auction start and end times, the performance of each tactic is a variable that can change depending on the conditions of the environment encountered and can be greatly influenced by the personal style of its user.

7.3 Reactive Rules (Jess Definition Rules)

Reactive rules are used for programming rule-based, reactive systems, which have the ability to detect events and respond to them automatically in a timely manner. Such systems are needed on the Web for bridging the gap between the existing, passive Web, where data sources can only be accessed to obtain information, and the dynamic Web, where data sources are enriched with reactive behaviour.

A category of reactive rules is the Jess definition rules. A Jess rule is something like an “if...then” statement in a procedural language, but it is not used in a procedural way. While “if...then” statements are executed at a specific time and in a specific order, according to how the programmer writes them, Jess rules are executed whenever their if parts (their left-hand-sides or LHS) are satisfied, given only that the rule engine is running. This makes Jess rules less deterministic than a typical procedural program. In other words, Jess rules are of the type: “on event, if condition is satisfied, then action is triggered”. The basic Jess rules designed in user tracking system are described below. They are targeting to cover two main theme categories, user feedback and user deviation. More information about Jess rules and Jess rule engine is presented in chapter 4.

7.3.1 Rules about the Ratio and the Number of Voters of the auctioneer

There are cases that the level of trust might be very low but if a great number of users have given a very good ratio to this auctioneer the C-agent might want to raise the level of trust.

Or contrary the group may have given a very good trust level but only a small number of users has voted this auctioneer and raised its ratio. This is usually a case where the auctioneer himself tries to raise his ratio. In these cases the C-agent has rules that lower the level of trust.

In the rules following we make use of the information we get from the Price agent we mentioned before.

```
(defrule fromlevel1tolevel2user

  ?auct <- (DefaultAuctionDetails (hasString ?name) (hasArtifact ?m) (hasStartingPrice ?v) (hasArtifactID ?tau){(hasTrust == "level1") && (hasRatio > 6) && (hasVoters > 100)} )

        => (printout t "Reactive Rule fromlevel1tolevel2user: " ?name crlf)

   (retract ?auct)
```

```
   (calling levelChange ?name level2 ?m)
)
```

Description **:** If the trust level of the auctioneer is level 1 but over 100 users have rated him with average rate over 6 then the fact representing the description of this auction is retracted and the jessfunction levelchange is called. In the function the javabean representing the auction Details of the specific auction is changed having now trust level 2 and the fact is asserted again.

```
(defrule fromlevel2tolevel3user
   ?auct <- (DefaultAuctionDetails (hasString ?name) (hasArtifact ?m) (hasStartingPrice ?v) (hasArtifactID ?tau){(hasTrust == "level2") && (hasRatio > 7) && (hasVoters > 200)} )

        => (printout t "Reactive Rule fromlevel2tolevel3user: " ?name crlf)
   (retract ?auct)
   (calling levelChange ?name level3 ?m)
)
```

Description **:** If the trust level of the auctioneer is level 2 but over 200 users have rated him with average rate over 7 then the fact representing the description of this auction is retracted and the jessfunction levelchange is called. In the function the javabean representing the auction Details of the specific auction is changed having now trust level 3 and the fact is asserted again.

```
(defrule fromlevel3tolevel4user
   ?auct <- (DefaultAuctionDetails (hasString ?name) (hasArtifact ?m) (hasStartingPrice ?v) (hasArtifactID ?tau){(hasTrust == "level3") && (hasRatio > 9) && (hasVoters > 200)} )

        => (printout t "Reactive Rule fromlevel3tolevel4user: " ?name crlf)
   (retract ?auct)
   (calling levelChange ?name level4 ?m)
)
```

Description **:** If the trust level of the auctioneer is level 3 but over 200 users have rated him with average rate over 9 then the fact representing the description of this auction is retracted and the jessfunction levelchange is called. In the function the javabean representing the auction Details of the specific auction is changed having now trust level 4 and the fact is asserted again.

```
(defrule fromlevel4tolevel5user

   ?auct <- (DefaultAuctionDetails (hasString ?name) (hasArtifact ?m) (hasStartingPrice ?v) (hasArtifactID ?tau){(hasTrust == "level4") && (hasRatio >= 10) && (hasVoters > 400)} )


        => (printout t "Reactive Rule fromlevel4tolevel5user: " ?name crlf)

   (retract ?auct)

   (calling levelChange ?name level5 ?m)

)
```

Description **:** If the trust level of the auctioneer is level 4 but over 400 users have rated him with average rate 10 then the fact representing the description of this auction is retracted and the jessfunction levelchange is called. In the function the javabean representing the auction Details of the specific auction is changed having now trust level 5 and the fact is asserted again.

```
(defrule fromlevel5tolevel4user

   ?auct <- (DefaultAuctionDetails (hasString ?name) (hasArtifact ?m) (hasStartingPrice ?v) (hasArtifactID ?tau){(hasTrust == "level4") && (hasRatio < 10) && (hasVoters < 400)} )


        => (printout t "Reactive Rule fromlevel5tolevel4user: " ?name crlf)

   (retract ?auct)

   (calling levelChange ?name level4 ?m)

)
```

Description **:** If the trust level of the auctioneer is level 5 but under 400 users have rated him with average rate below 10 then the fact representing the description of this auction is retracted and the jessfunction levelchange is called. In the function the javabean representing the auction Details of the specific auction is changed having now trust level 4 and the fact is asserted again.

```
(defrule fromlevel4tolevel3user

?auct <- (DefaultAuctionDetails (hasString ?name) (hasArtifact ?m) (hasStartingPrice ?v) (hasArtifactID ?tau){(hasTrust == "level3") && (hasRatio < 9) && (hasVoters < 200)} )
```

```
        => (printout t "Reactive Rule fromlevel4tolevel3user: " ?name crlf)
   (retract ?auct)
   (calling levelChange ?name level3 ?m)
)
```

Description **:** If the trust level of the auctioneer is level 4 but under 200 users have rated him with average rate below 9 then the fact representing the description of this auction is retracted and the jessfunction levelchange is called. In the function the javabean representing the auction Details of the specific auction is changed having now trust level 3and the fact is asserted again.

```
(defrule fromlevel3tolevel2user
?auct <- (DefaultAuctionDetails (hasString ?name) (hasArtifact ?m) (hasStartingPrice ?v) (hasArtifactID ?tau){(hasTrust == "level3") && (hasRatio < 7) && (hasVoters < 200)} )

        => (printout t "Reactive Rule fromlevel3tolevel2user: " ?name crlf)
   (retract ?auct)
   (calling levelChange ?name level2 ?m)
)
```

Description **:** If the trust level of the auctioneer is level 3 but under 200 users have rated him with average rate below 7 then the fact representing the description of this auction is retracted and the jessfunction levelchange is called. In the function the javabean representing the auction Details of the specific auction is changed having now trust level 2 and the fact is asserted again.

```
(defrule fromlevel2tolevel1user
?auct <- (DefaultAuctionDetails (hasString ?name) (hasArtifact ?m) (hasStartingPrice ?v) (hasArtifactID ?tau){(hasTrust == "level3") && (hasRatio < 6) && (hasVoters < 100)} )

        => (printout t "Reactive Rule fromlevel2tolevel1user: " ?name crlf)
   (retract ?auct)
```

```
   (calling levelChange ?name level1 ?m)
)
```

Description **:** If the trust level of the auctioneer is level 3 but under 100 users have rated him with average rate below 6 then the fact representing the description of this auction is retracted and the jessfunction levelchange is called. In the function the javabean representing the auction Details of the specific auction is changed having now trust level 1 and the fact is asserted again.

7.3.2 Rules about the AvgSalingPrice

Another parameter that must always affect the trust level of an auctionner is the average selling price of the items he auctions. That is because the agent may have great trust level but for selling cheap things and if a user wants to buy something expensive he must be cautious. Anyone can understand that if a user wants to buy a laptop for instance he is going to be willing to give more money to a user with trust level 3 and avg sales price 2000 euro than a user with trust level 5 and avg sales price 20 euro.

```
(defrule fromlevel5tolevel3

?auct <- (DefaultAuctionDetails (hasString ?name) (hasArtifact ?m)
{
      (hasTrust == "level5") && ( hasAVGPrice < (* (/ 20 100) hasPrice))}  )
         => (printout t "Reactive Rule fromlevel5tolevel3: " ?name crlf)
   (retract ?auct)
   (calling levelChange ?name level3 ?m)
)
```

Description **:** If the trust level of the auctioneer is level 5 but its average sales price is under 20% of the average sale price of the product he is selling then the fact representing the description of this auction is retracted and the jessfunction levelchange is called. In the function the javabean representing the auction Details of the specific auction is changed having now trust level 3 and the fact is asserted again.

```
(defrule apolevel4selevel3

?auct <- (DefaultAuctionDetails (hasString ?name) (hasArtifact ?m)
 {
      (hasTrust == "level4") && ( hasAVGPrice < (* (/ 60 100) hasPrice))}  )
         => (printout t "Reactive Rule apolevel3selevel4: " ?name crlf)
   (retract ?auct)
   (calling levelChange ?name level3 ?m)
)
```

Description **:** If the trust level of the auctioneer is level 4 but its average sales price is under 60% of the average sale price of the product he is selling then the fact representing the description of this auction is retracted and the jessfunction levelchange is called. In the function the javabean representing the auction Details of the specific auction is changed having now trust level 3 and the fact is asserted again.

```
(defrule apolevel3selevel4
         ?auct <- (DefaultAuctionDetails (hasString ?name) (hasArtifact ?m)
{
 (hasTrust == "level3") && ( hasAVGPrice > (* (/ 500 100) hasPrice))}  )
         => (printout t "Reactive Rule apolevel3selevel4: " ?name crlf)
   (retract ?auct)
   (calling levelChange ?name level4 ?m)
)
```

Description **:** If the trust level of the auctioneer is level 4 but its average sales price is over 500% of the average sale price of the product he is selling then the fact representing the description of this auction is retracted and the jessfunction levelchange is called. In the function the javabean representing the auction Details of the specific auction is changed having now trust level 4 and the fact is asserted again.

```
(defrule apolevel2selevel3


   "Testing the rule."

                ?auct <- (DefaultAuctionDetails (hasString ?name)  (hasArtifact ?m)
{
(hasTrust == "level2") && ( hasAVGPrice > (* (/ 200 100) hasPrice))}  )
  => (printout t "Reactive Rule apolevel2selevel3: " ?name crlf)
   (retract ?auct)
   (calling levelChange ?name level3 ?m)
)
```

Description **:** If the trust level of the auctioneer is level 2 but its average sales price is over 200% of the average sale price of the product he is selling then the fact representing the description of this auction is retracted and the jessfunction levelchange is called. In the function the javabean representing the auction Details of the specific auction is changed having now trust level 3 and the fact is asserted again.

7.3.3 Customer's Preferences

Another function of our system that we described earlier is the ability of the user to give his preferences about the characteristics of the item he is going to buy. After creating all the auctions Javabeans using the last ontology given the C-agent with the help of OWL API extracts all the artifacts properties and creates a Pop up window like the one in the figure. Next to each property there is a text field the user can fill with the desired values (by default it contains the type of the property). Then the user must select if it is optional or not that this property should match that.

- If it is not optional he must select the tick box
- If it is optional he must give a value in the last column showing how important for him it is that the property matches his preference.

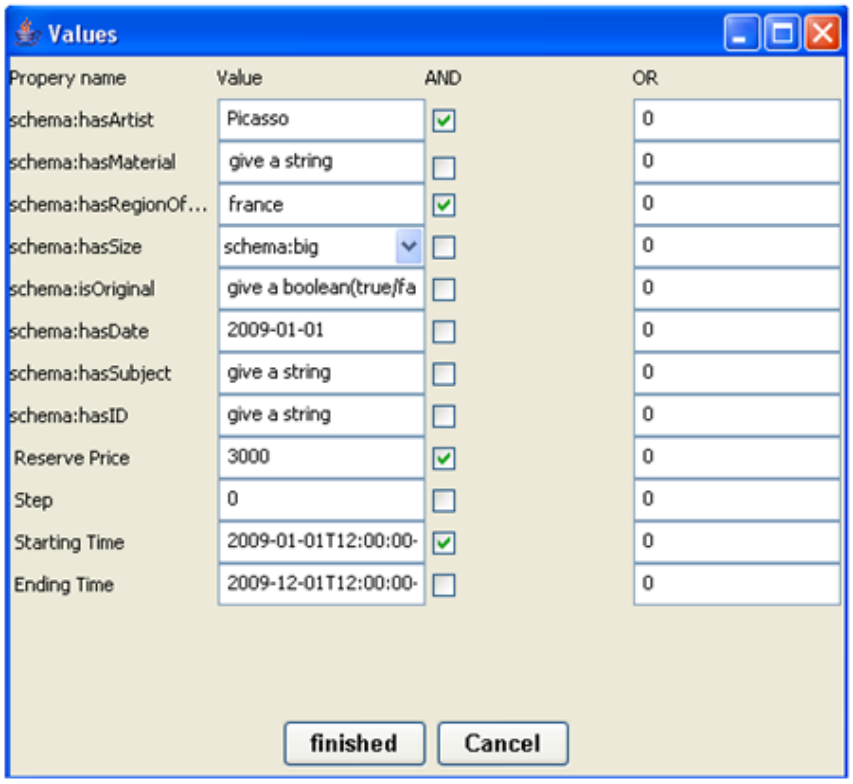


Figure 35 Example of rule when only obligatory fields are filled

(defrule elegxosprotimisewn

(DefaultAuctionDetails (hasString ?name) (hasArtifact ?m) (hasArtifactID ?tau) {

(hasStartingTime >= 1.238576400953E12) && (hasStartingPrice <= 3000) })

(DefaultSculpture (hasID ?tau1) {hasID == ?tau} {(hasArtist == "Picasso") && (hasRegionOfOrigin == "france")})

=> (printout t "Reactive Rule elegxosprotimisewn: " ?name crlf) (calling elegxosprotimisewn ?name ?m))

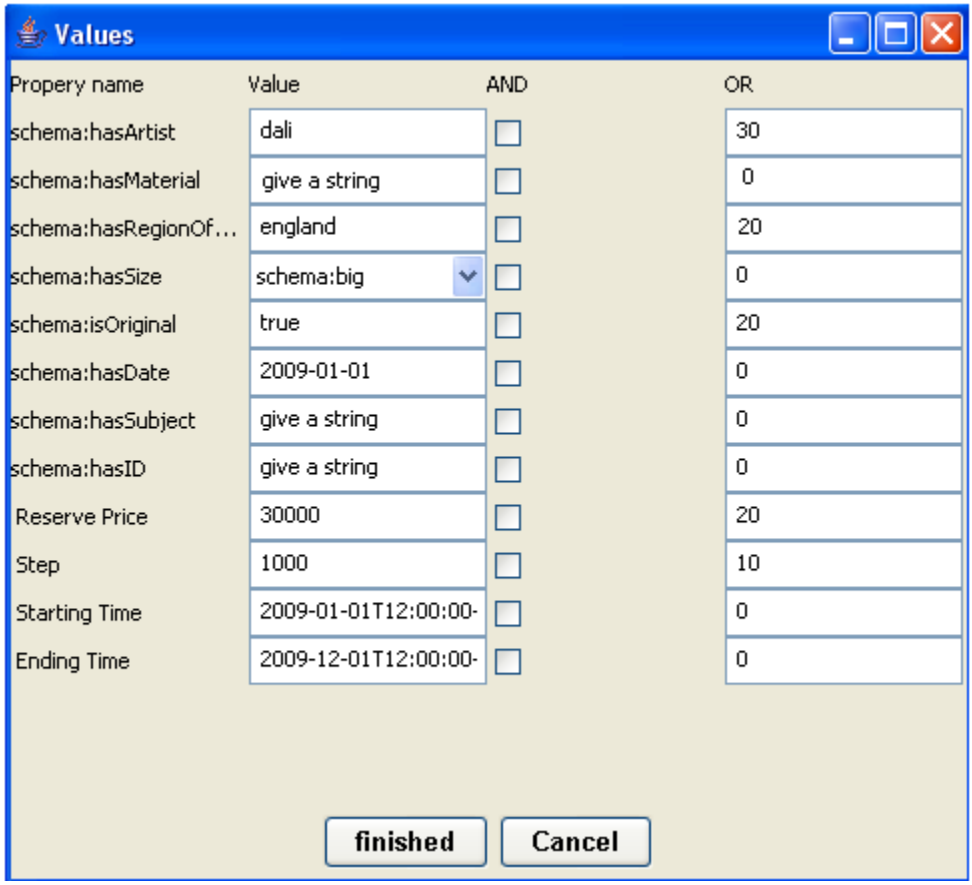


Figure 36 Example of rule when only optional fields are used

defrule elegxosprotimisewn

(DefaultAuctionDetails (hasString ?name) (hasArtifact ?m) (hasArtifactID ?tau) { (hasStartingPrice <= 30000) || (hasStep <= 1000) })

(DefaultSculpture (hasID ?tau1) {hasID == ?tau} {(hasArtist == "dali") || (hasRegionOfOrigin == "england") || (isOriginal == "true")})

=> (printout t "Reactive Rule elegxosprotimisewn: " ?name crlf) (calling elegxosprotimisewn ?name ?m))|

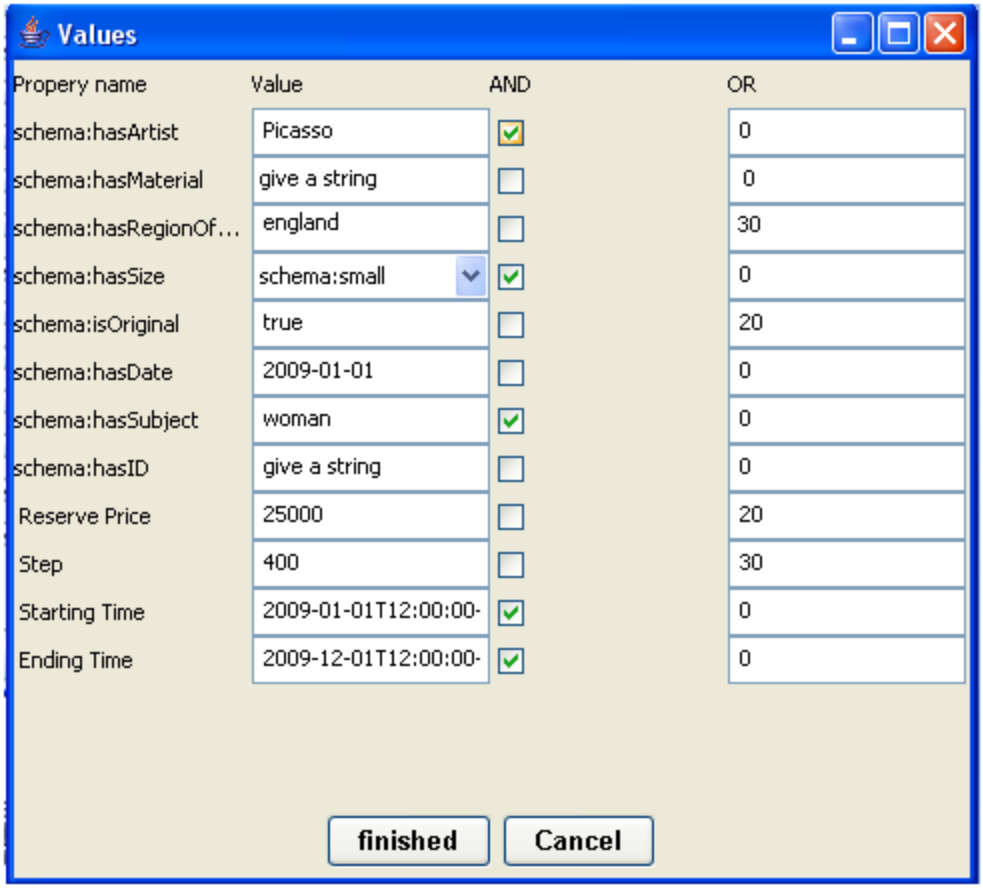


Figure 37 Example of rule when both are used

(defrule elegxosprotimisewn

(DefaultAuctionDetails (hasString ?name) (hasArtifact ?m) (hasArtifactID ?tau) {

(hasEndingTime <= 1.259661600171E12) && (hasStartingTime >=

1.230804000171E12) } { *(hasStartingPrice <= 25000)* || *(hasStep <= 400)* })

(DefaultSculpture (hasID ?tau1) {hasID == ?tau} {(hasArtist == "Picasso") && (hasSize == "small") && (hasSubject == "woman")}

{(hasRegionOfOrigin == "England")||(isOriginal == "true")})

=> (printout t "Reactive Rule elegxosprotimisewn: " ?name crlf) (calling elegxosprotimisewn ?name ?m))

The way the matching result is counted examples is shown in the following examples

- First example

When there are only optional fields

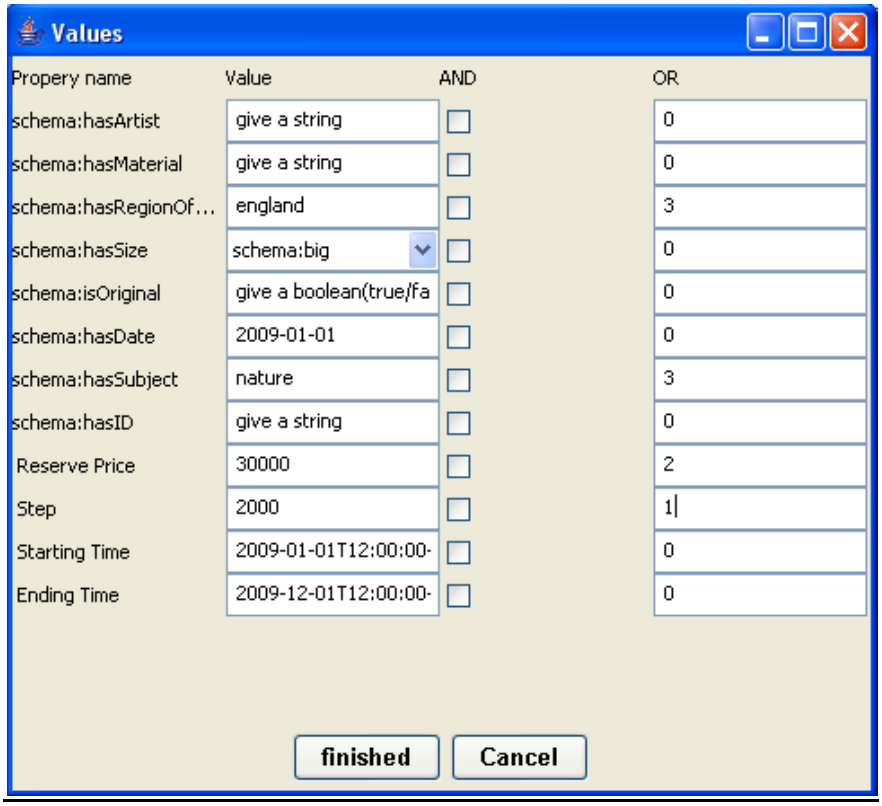


Figure 38 When there are only optional fields

(defrule elegxosprotimisewn

(DefaultAuctionDetails (hasString ?name) (hasArtifact ?m) (hasArtifactID ?tau) { (hasStartingPrice <= 3000) || (hasStep <= 200) })

(DefaultSculpture (hasID ?tau1) {hasID == ?tau} {(hasRegionOfOrigin == "england")||(hasSubject == "nature")})

⇒ (printout t "Reactive Rule elegxosprotimisewn: " ?name crlf) (calling elegxosprotimisewn ?name ?m))

hasStartingPrice➔20%

hasStep➔10%

hasRegionOfOrigin➔30%

hasSubject➔40%

In order to find the matching percent of each property that is optional the following action are taken:

We first find the sum of the values entered (in the example 2+1+4+3=10)

Then we find each ones percent by using the type value/sum*100 where value is the value entered in the last coloumn and sum the result we found before(in the example for hasStep 1/10*100=10)

If an artifact satisfies the first two fields (hasStartingPrice and hasStep) the matching result is 30%

If it satisfies the first three fields (hasStartingPrice , hasRegionOfOrigin and hasStep)the matching result is 70%

So for each matching combination the results is the sum of the matching percent of the fields included in the combination.

- Example of rule when both optional and non optional fields are used

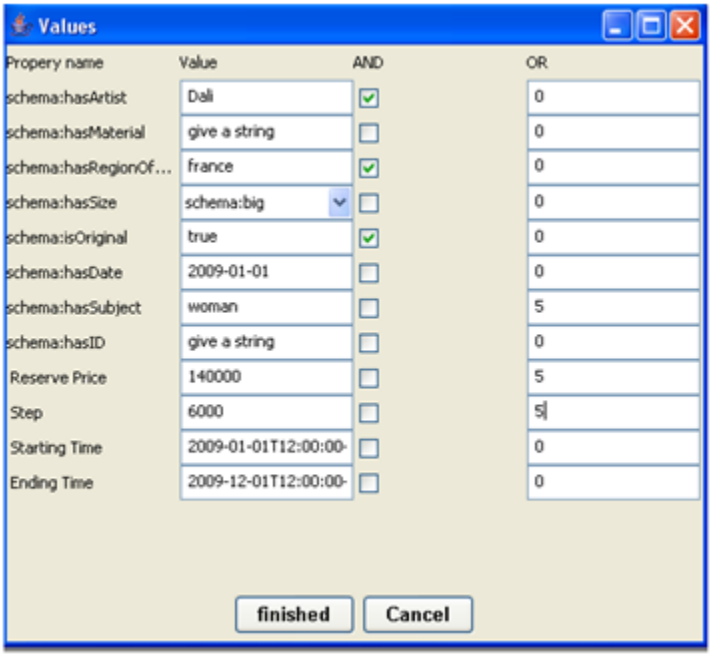


Figure 39 when both optional and non optional fields are used

(defrule elegxosprotimisewn

(DefaultAuctionDetails (hasString ?name) (hasArtifact ?m) (hasArtifactID ?tau) { (hasStartingPrice <= 140000) || (hasStep <= 6000) })

(DefaultSculpture (hasID ?tau1) {hasID == ?tau} {(hasArtist == "Dali")&&(hasRegionOfOrigin == "France")&&(isOriginal == "true")} {(hasSubject == "woman")})

=> (printout t "Reactive Rule elegxosprotimisewn: " ?name crlf) (calling elegxosprotimisewn ?name ?m))

The exact match has value 100 and from this 100% each non optional value get 1 share and all the optional parts together 1 share so

First we find the number X of the non optional parts (in our examples 3) and if there exist optional parts increase X by one (in our examples 3+1)

Then we divide 100/X in order to find the part of the 100% each part gets(in our examples 100/4=25)

If there exist optional parts their percent that we find in the previous step is shared to each one using the method we described in the previous example.

So in our example

hasRegionOfOrigin➔25%

isOriginal➔25%

hasArtist➔25%

hasSubject➔5/15*25%

hasStartingPrice➔5/15*25%

hasStep➔5/15*25%

If an artifacts matches in hasRegionOfOrigin and isOriginal it gets 50%

If it matches in hasRegionOfOrigin, has Artist and hasStep it gets (25+25+5/15*25) %

etc

7.3.4 Shilling rules

To manage to implement the shilling logic described in paragraph 5.2 we created the following JESS templates

```
(deftemplate SameAuctionItems
        (slot Item) (slot similar) (slot diafora))
```

This template keeps pairs of items that are the same

Item : the URL of an auction

Similar: the URL of an auction that sells the exactly same item as the previous one

Diafora: the time difference (in seconds) between the ending times of the two auctions. It is a positive number if similar ends after item and is a negative number if item ends after similar.

```
(deftemplate Bid
   (slot Client) (slot AuctionUrl) (slot Time) (slot Price)
   (slot Sender) (slot Active)   )
```

For each bid notification the C-agent receives from the CL-agents he stores

Client: the name of the winning C-agent for that auction until now

AuctionUrl: the URL of the auction

Time: the time the bid notification was sent

Price: the current time of the auction

Sender: the clone that sent the notification

Active: the URL of the client's active auction

```
(deftemplate SleepReason
    (slot AuctionUrl) (slot Reason)  )
```

If a C-agent has gone into monitoring mode into monitoring in the auction he is active we keep here the URL of the Active auction(slot AuctionUrl) and the URL of the auction that is the reason(slot Reason) he entered monitor mode.

```
(deftemplate Participates
    (slot AuctionUrl)  (slot Agent) )
```

Here we keep information about which C-agents participate in each one of the auction we chose to monitor.

```
(deftemplate FakeBid
   (slot Client) (slot AuctionUrl) (slot Times))
```

This template keeps the number of shilling bids an agent has done in an auction.

The rules we use in order to keep up to date the rule engine instance that is responsible for keeping track of the bids exchanged and monitoring the CL-agents and the A-agents auctions they participate in are the following ones.

```
(defrule DeleteOlder
  ?temp <- (Bid
        (AuctionUrl ?au1) (Time ?t1) (Sender ?sen1)
   )
   (Bid
        (AuctionUrl ?au2) (Time ?t2) (Sender ?sen2)
      {AuctionUrl == ?au1 && Sender == ?sen1 && (Time > ?t1) }

   )
   =>
```

```
    (printout t "Reactive Rule DeleteOlder: " crlf)
    (retract ?temp)
)
```

For each Auction we keep only the last bid information. Previous bids are retracted as they are no longer of any use since they are not valid any more

```
(defrule foundBetterPrice

     (SameAuctionItems
       (Item ?it1) (diafora ?diaf1) (similar ?sim1)
       { Item == ?au1 && diafora < 0 }
     )

    (Bid
       (AuctionUrl ?au1) (Time ?t1)
       (Price ?p1) (Sender ?sen1) (Active ?act1)
       {AuctionUrl <> Active  }
     )
    (Bid
       (AuctionUrl ?au2) (Time ?t2)
       (Price ?p2) (Sender ?sen2) (Active ?act2)
       {(AuctionUrl == ?sim1) && (AuctionUrl == Active) && (Price >       ?p1)  &&
(Time > ?t1) }
       )

    =>
     (printout t "Reactive Rule foundBetterPrice: " crlf)
    (assert (SleepReason (AuctionUrl ?au2) (Reason ?au1)))
```

```
   (send (assert (ACLMessage (communicative-act INFORM) (receiver ?sen2)
(content "sleep") )))

   )
```

If the C-agent takes part in two auctions that sell the exact same item and the last bid he received from the auction that is not his active one right now and ends later than the active one has lower price than the last bid he received from his active one the C-agents keeps monitoring the active auction but does not send bids for the time the other auctions price is better. He manages that by informing the CL-agent of the active mode to go to monitor mode. Also the C-agent asserts a fact containing the active auctions URL and the auction that made him stop bidding URL.

```
(defrule ReactivateAuction

     ?temp<-(SleepReason

                (AuctionUrl ?au3) (Reason ?au4)

        )

   (Bid

      (Client ?cl1) (AuctionUrl ?au1) (Time ?t1)

      (Price ?p1) (Sender ?sen1) (Active ?act1) (Mode ?mod1)

      {AuctionUrl != Active && AuctionUrl == ?au4}

    )


   (SameAuctionItems

      (Item ?it1) (diafora ?diaf1) (similar ?sim1)

      { Item == ?au1 && diafora < 0 }

    )



    (Bid

      (Client ?cl2) (AuctionUrl ?au2) (Time ?t2)

      (Price ?p2) (Sender ?sen2) (Active ?act2)
```

```
      {AuctionUrl == ?au3 && AuctionUrl == ?sim1 && AuctionUrl == Active &&
Price < ?p1  }

      )


   =>

   (printout t "Reactive Rule ReactivateAuction: " crlf)

   (retract ?temp)

   (send (assert (ACLMessage (communicative-act PROPOSE) (receiver ?sen2)
(content "wake") )))

   )
```

This is the rule that reactivates the auction when the other auction's price becomes greater than the auction that the C-agent is taking place at the moment.

```
(defrule ShillBid
        (Participates (AuctionUrl ?au4) (Agent ?ag1))                    (1)


   (Participates (AuctionUrl ?au5) (Agent ?ag2)                          (2)
      {Agent == ?ag1 && AuctionUrl != ?au4}

      )


  ?temp2<-  (Bid                                                         (3)
      (Client ?cl1) (AuctionUrl ?au1) (Time ?t1)

      (Price ?p1) (Sender ?sen1) (Active ?act1) (Mode ?mod1)

      {AuctionUrl == ?au4 && Client == ?ag1 }

    )


   (SameAuctionItems                                                     (4)
```

```
      (Item ?it1) (diafora ?diaf1) (similar ?sim1)
      { Item == ?au1 && diafora > 0 }
    )

    (Bid                                                    (5)
      (Client ?cl2) (AuctionUrl ?au2) (Time ?t2)
      (Price ?p2) (Sender ?sen2) (Active ?act2) (Mode ?mod2)
      {AuctionUrl == ?sim1 && AuctionUrl == ?au5 && Price < ?p1  && Time < ?t1
}

      )

  ?out<-( FakeBid                                           (6)
  ( Client ?fb1) (AuctionUrl ?fb2) (Times ?fb3)
      {Client == ?cl1 && AuctionUrl == ?au1}
      )

   =>
   (printout t "Reactive Rule ShillBid: " crlf)
   (retract ?out)
   (retract ?temp2)
   (calling ShillBid ?fb1 ?fb2 ?fb3)



   )
```

The C-agent takes part in two auctions ((1), (2)) and these two auctions offer the two item with the same characteristics(4). The C-agent send a bid (4) to an auction (3) but there exists an auction that will end later and has a better price (5) then the bid the C-agent made is considered a shilling bid. The shilling bid is

removed from the knowledge base and the number of shilling bids (6) the C-agent has made in this auction is changed.

7.4 Auction Implementation

The implementation of the auction protocols follows formalisms of traditional auction regulations and principles. Concerning the English auction, the auctioneer accepts offers greater than the running Maximum Offer and broadcasts the new bid to all participants along with information about the bidder. After the termination of the auction, only one of the participants wins at a price equal to the value of the maximum proposed bid. The Vickrey auction on the other hand is different, because all participants place their bids, but none of them is publicly announced. Only the auctioneer knows the candidate offers and informs the participants whenever a new offer has been accepted. After the end of the session, the participant that has placed the highest offer wins and pays the price of the second highest bid. No other information is broadly announced.

Another different approach from the SeMPHoNIA is that there is no distinction between C-agent and statistical Agent. In SeMPHoNIA when an agent wanted to take part in many auction and win only one of them only one clone was active at the time and the others were just waiting. But in our approach we need to monitor all auctions for shilling behavior so the Clone does both jobs and monitor and bid whenever needed. The creation of two deferent agents would result in two clone agents for each auction and would load the network with a number of agents double than now.

Concerning bid acceptance and withdrawal we followed the same tactic followed in SeMPHoNIA auctions. All bidders are allowed to withdraw their offers taking into account an important detail. A withdrawal of the winning bid during the progress of an auction will provisionally not be accepted, until another bidder matches the price of that offer and releases it. This policy has been implemented in other auction applications as well ([24], [23]).

In cases where the information is private (for example, offers in Vickrey auctions), when we create messages that contain fields about those information the corresponding slots are filled with the *null* value. Thus, all registration data required by agents to begin interacting in an auction are provided by an auction-type-independent element promoting the uniformity of the platform.

7.4 .1The strategy followed in the bidding process

The bidding strategy of our agent has three different phases

PHASE 1 SETTING THE MAX BID

After the trust level of the auctionner is set the C-Agent decides the maximum bid he is willing to give for the specific auction taking under consideration the following three parameters

The trust level A.

The maximum price the Clinet said he is willing to pay B.

The average price in which items like the one auctioned are sold C.

The matching result of the users preferences and the auctioned item's characteristics D%

In case of low level trust (level 1 or level 2) the agent find which is the smallest between B and C so

He chooses to do this because the auctioneer is not trusted and so if he was willing to give a Max price much bigger than the average price he used to give this should not happen. Instead in this case the Max price is replaced with the average price this type of items are sold.

If(B>C)

X=C

Else

X=B

And then

If A=1

MAXBID= X*55/100 + D/10*X/100;

So the MAXBID will be from 55% to 65% of X and this depends on the matching result

If A=2

MAXBID= X*75/100 + D/10*X/100;

So the MAXBID will be from 75% to 85% of X and this depends on the matching result

In case of levels trust 3, 4, 5 the agent is willing to give the MAXBID that is counted from the following rules

If A=3

MAXBID= B*75/100 + D/10*B/100;

So the MAXBID will be from 75% to 85% of the maximum price the Client said he is willing to pay and this depends on the matching result

If A=4

MAXBID= B*80/100 + D/10*B/100;

So the MAXBID will be from 80% to 90% of the maximum price the Client said he is willing to pay and this depends on the matching result

If A=5

MAXBID= B*90/100 + D/10*B/100;

So the MAXBID will be from 90% to 100% of the maximum price the Client said he is willing to pay and this depends on the matching result

PHASE 2 The Remaining Time Tactic

This tactic [90] determines the recommended bid value based on the amount of time remaining for the agent. Assume that the agent is bidding at time $0 \leq t \leq t_{max}$. The agent bids closer to p_r as t approaches t_{max}, and it eventually bids at p_r when $t = t_{max}$. To calculate the bid value at time t, the following expression is used:

$f_{rt} = a_{rt}(t)\, p_r$

where $a_{rt}(t)$ is a polynomial function of the form:

$$a_{rt}(t) = k_{rt} + (1 - k_{rt})\left(\frac{t}{t_{max}}\right)^{1/\beta}$$

k_{rt} is a constant that when multiplied by the size of the interval determines the value of the starting bid of the agent in any auction. By varying the value of $a_{rt}(t)$, a wide range of time dependent functions can be defined from those that start bidding near p_r quickly, to those that only bid near p_r right at the end, to all possibilities in between. The only condition is that $0 \leq a_{rt}(t) \leq 1$, $a_{rt}(0) = k_{rt}$, $a_{rt}(t_{max}) = 1$, and $0 \leq k_{rt} \leq 1$.

Using the polynomial function defined earlier, different shapes of curves can be obtained by varying the values of β . This represents an infinite number of possible tactics, one for each value of β . In this tactic, β is drawn from $\mathcal{R}^+$, where 1/ 200 ≤ β ≤ 1000 . When β < 1 , the tactic maintains a low bid value until the deadline is almost reached, where this tactic concedes by suggesting the reservation price as the recommended bid value. The other extreme is when β > 1; the tactic starts with a bid value close to the reservation price and quickly reaches the reservation value long before the deadline is reached.

When The C-agent creates the CL-agents according to their trust level he sets the value of β making the agent more aggressive when the A-agent is trusted and more conservative when the A-agent is untrusted.

PHASE 3 The Remaining Auctions Tactic

This tactic is broadly similar to the remaining time tactic; the agent bids closer to p_r as the number of remaining auctions approaches 0 (since it is running out of opportunities to purchase the desired good) [90].

f_{ra} has the same form as f_{rt} and a_{ra} is defined as follows:

$$a_{ra} = k_{ra} + (1 - k_{ra})\left(\frac{c(t)}{|\Lambda|}\right)^{1/\beta}$$

Most of these terms are similar to a_{rt} , the only difference being that $c(t)$ is the list of all the auctions that have been completed between time 0 and time t . β is again drawn

from R^+ , where 1/ 200 ≤ β≤ 1000 .

Selecting Potential Auctions and the Target Auction

The agent participate in an auction if and only if it is not holding the highest bid in another English auction, or it has not placed a bid in a Vickrey auction. This is to ensure that the agent does not acquire more than one of the target items.

The agent selects the potential auctions by considering values for the remaining time for each active auction. The agent's new bid value for an auction is the current bid plus the bid increment. The choice to follow the auctions according to their end time is taken for the following reason. If the agent currently holds a bid in an English auction that still has a long time to complete, it will not be able to participate in other auctions until it loses out to another bidder or until

the auction terminates and so he misses the chance to participate in auctions that end before that. There are two potential outcomes when an auction terminates; the agent loses out to another bidder or the agent wins. If the first occurred and we had followed a different tactic, the agent would have lost out in that it wasted a lot of time in one auction, thus reducing its chances of participating in other auctions but with our strategy he knows he will take part in all the auctions. The potential Vickrey auctions in which the agent may bid are those that are terminating at the current time and the agent's bid value is its current maximum bid value. This selection of timing is based on the same reasoning as that of the English auction.

So if there is only one auction in the potential auction list, that auction is picked as the target auction. If there are multiple auctions, the agent calculates the remaining time for each of these potential auctions. The auction which ends earlier for the agent's bid value is picked as the target auction. Here, the probability of winning is highly dependent on the type of auction and the agent's bid value.

In the end all the auctions the user selected will at some time be active and none of them will be overlooked by the C-agent. But as we said what makes the difference is the trust level of each auction and the matching result that affect the max bid the agent is willing to give.

After explaining, in this chapter, certain implementation aspects of our platform, we are going to present in the next two chapters some examples and results of performance evaluation carried out through numerous real-condition test cases. We compare the performance of different methods and, also, attempt an interpretation of the findings obtained.

8 Platform scenarios

The purpose of the first scenarios is to show how the system works when users take part in multiple auctions and how the agents move from auction to auction. The second scenario shows the reaction of the C-Agent when it find a shill agent taking part in an auction.

8.1 Agents Participating in many auctions

In order to show how the agents work when taking part in many auctions we initiated 3 A-agents mary, manolis and petros. Mary ends first then manolis and finally petros.

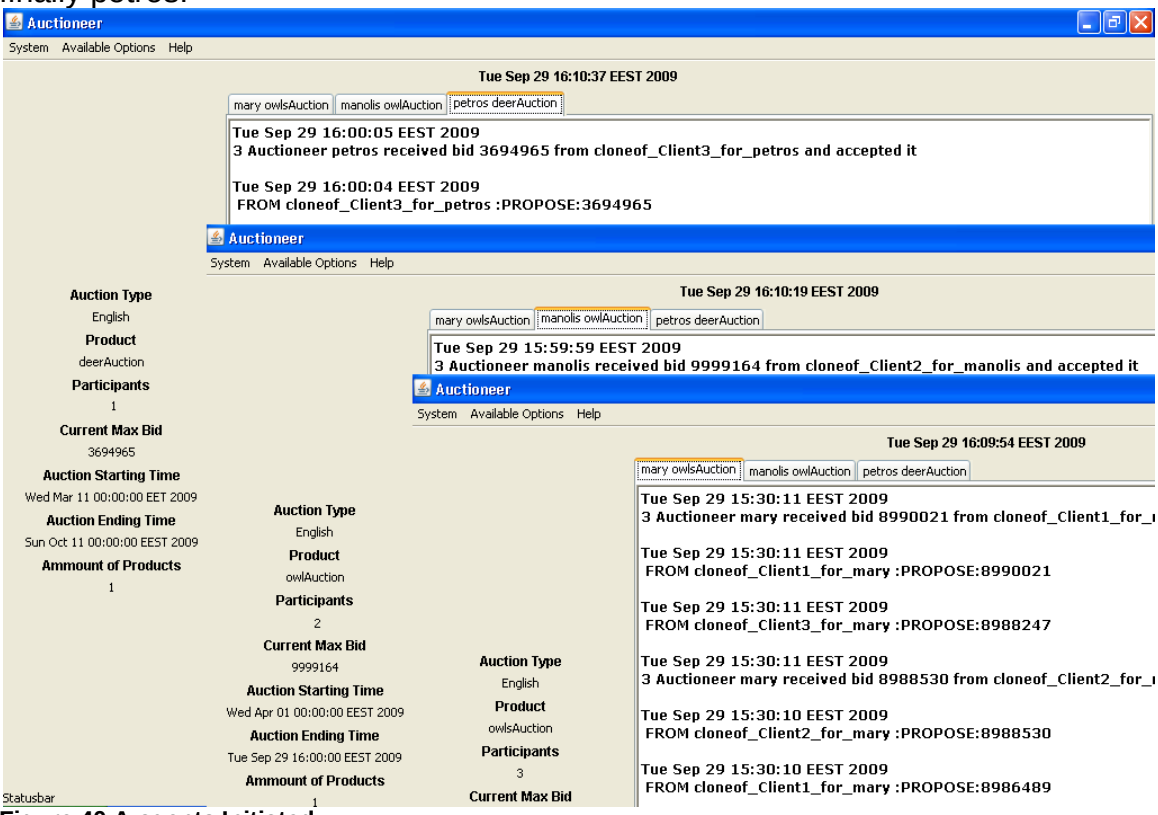


Figure 40 A-agents Initiated

Then we created Client1, Client2 and Client3 all taking part in the tree Auctions.

The agents begin being active in the A-agent mary until its end time. When mary ends it notifies Client1's clone that he won and the other two Clients' that they lost and they inform the C-agents that created them.

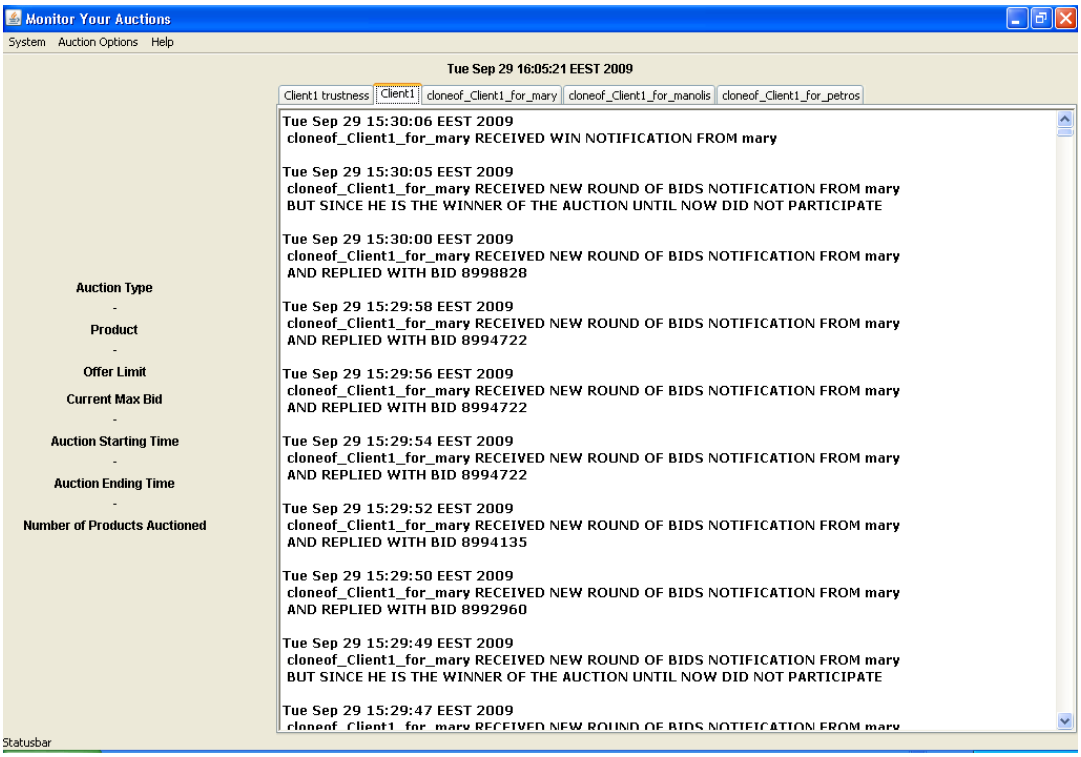


Figure 41 Client1 wins in mary

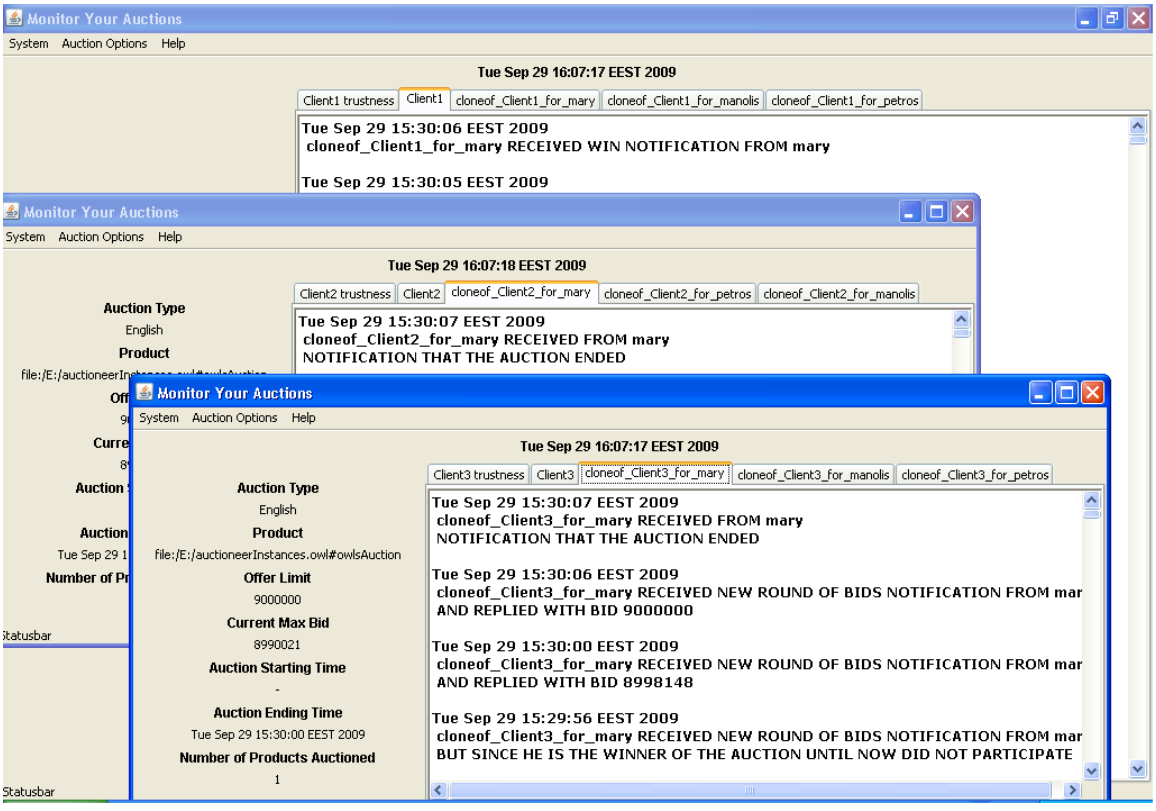


Figure 42 Notifications sent when mary ends

After mary ends Client2 and Client three become active in the next auction which is the auction of A-agent manolis. When manolis A-agent's ending time arrives it notifies Client2 that he is the winner and Client3 that the auction ended.

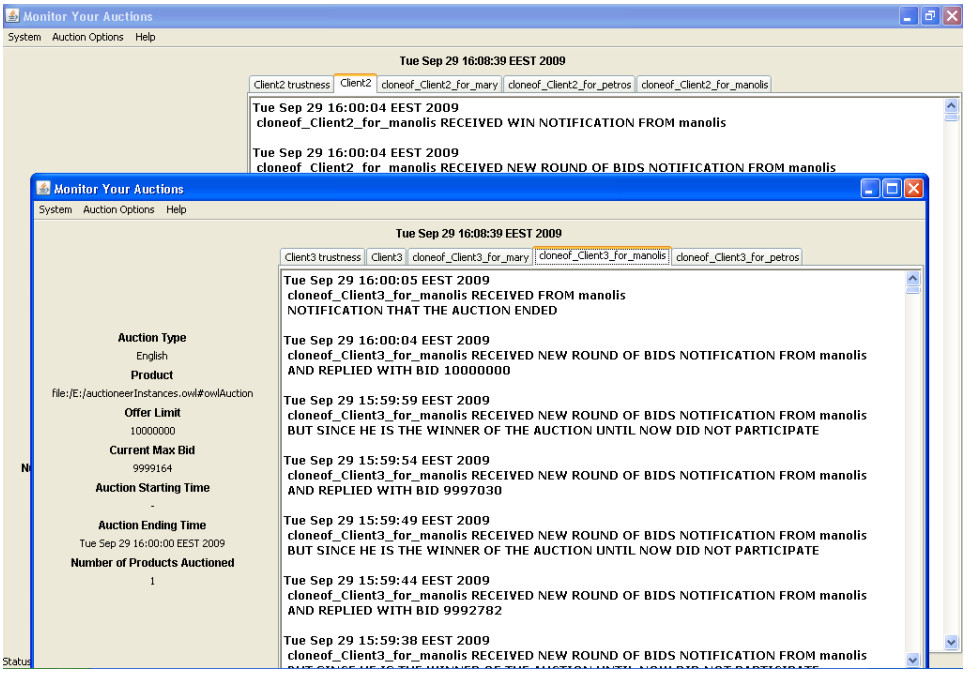


Figure 43 manolis ends

After manolis's end Client3 becomes active in the final Auction of A-agent petros.

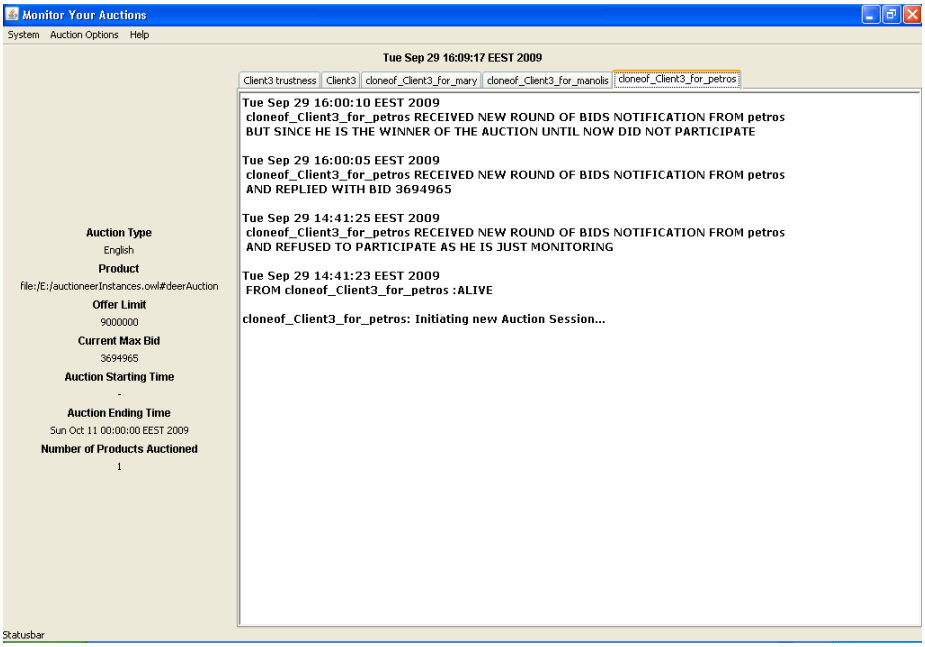


Figure 44 Client3 becomes active in A-agent petros

Finally we show some figures of the charts created during the example's progress

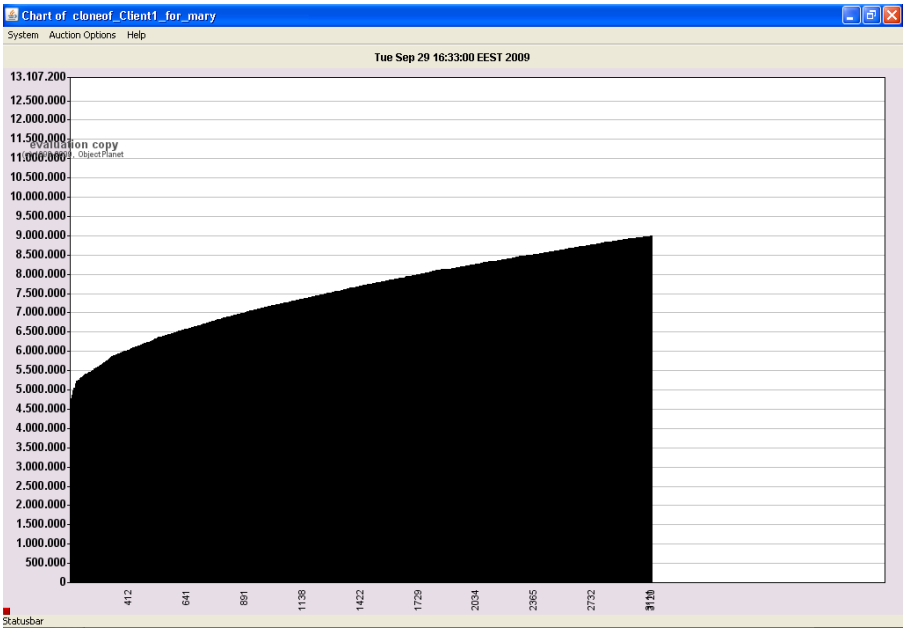


Figure 45 chart of Clien1's clone for mary

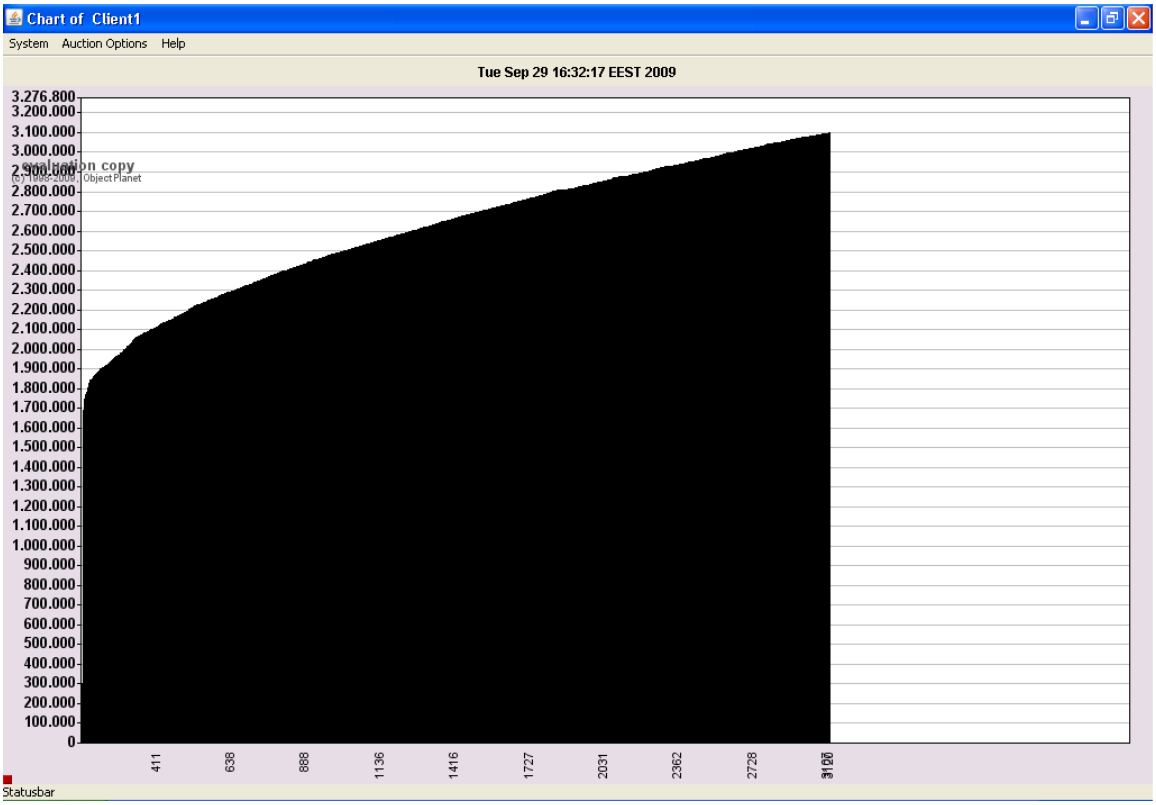


Figure 46 Chart of Client1 showing the Average selling price of all the auctions he takes part in

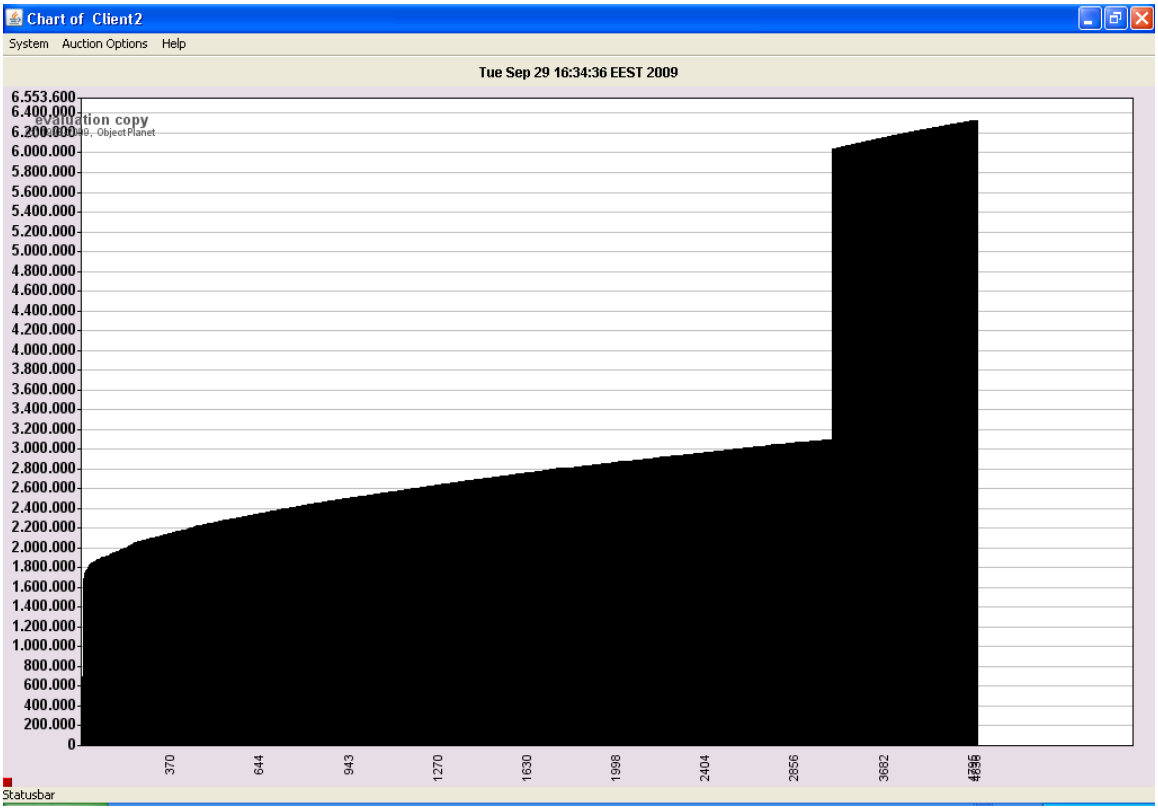


Figure 47 chart of Client2 showing the Average selling price of all the auctions he takes part in

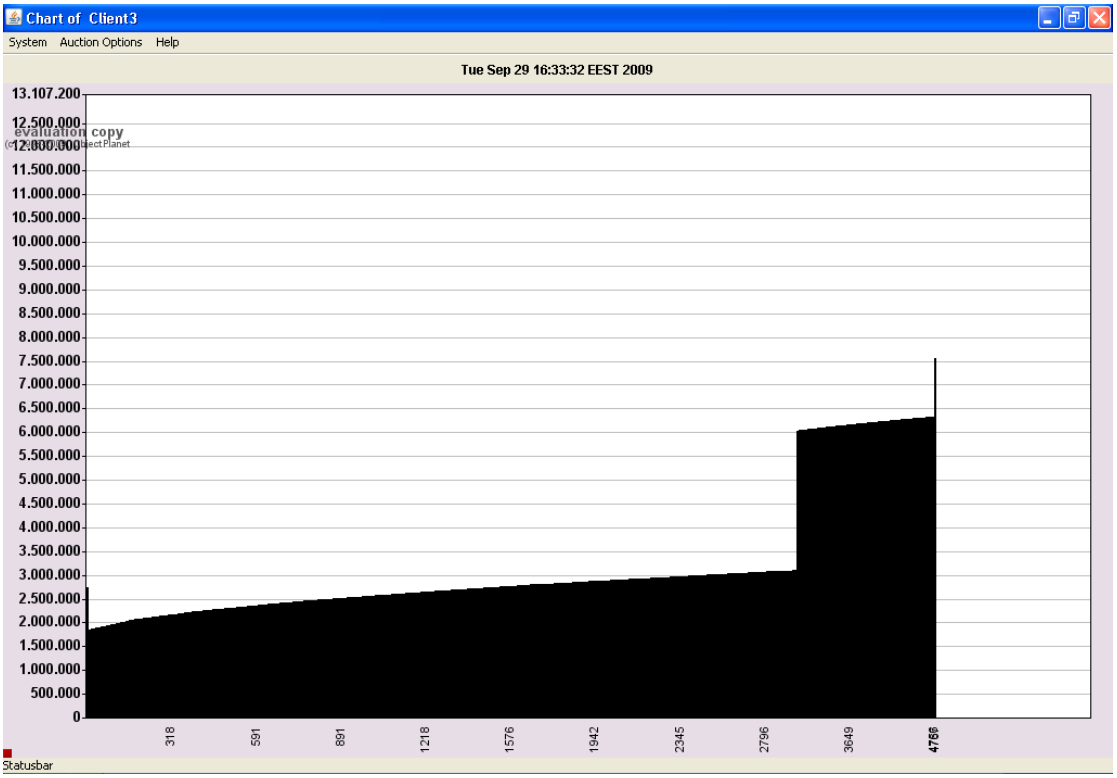


Figure 48 chart of Client3 showing the Average selling price of all the auctions he takes part in

8.2 Auction with shilling – Agent in it

In order to show how a C-agent reacts when it finds shilling behavior we executed the following scenario.
We initiated two agents that sell the same item

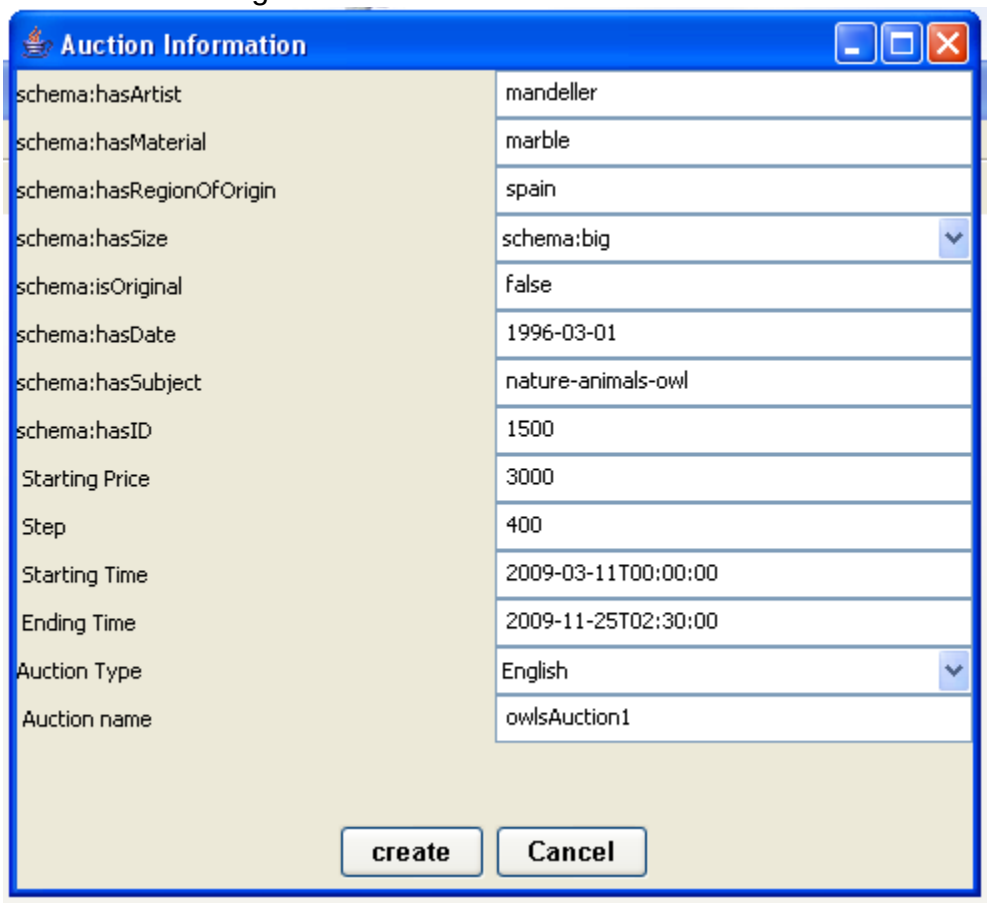


Figure 49 Action Details for A-agent mary1

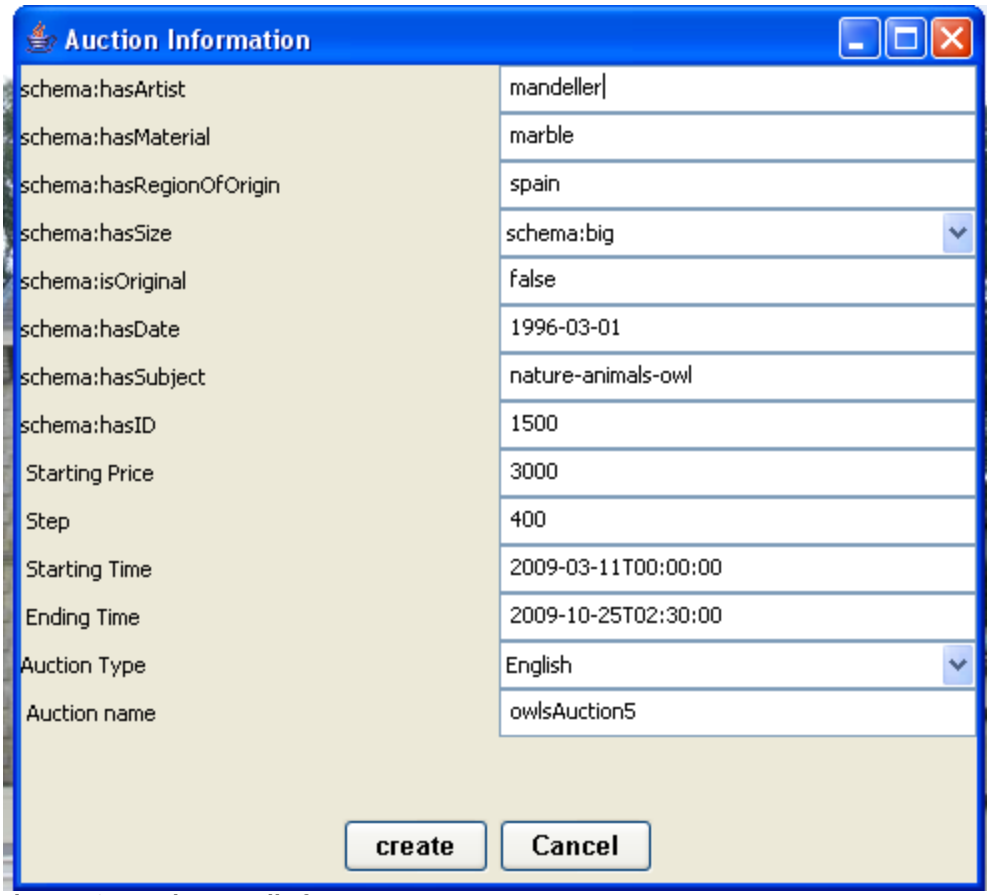


Figure 50 Auction Details for A-agent mary5

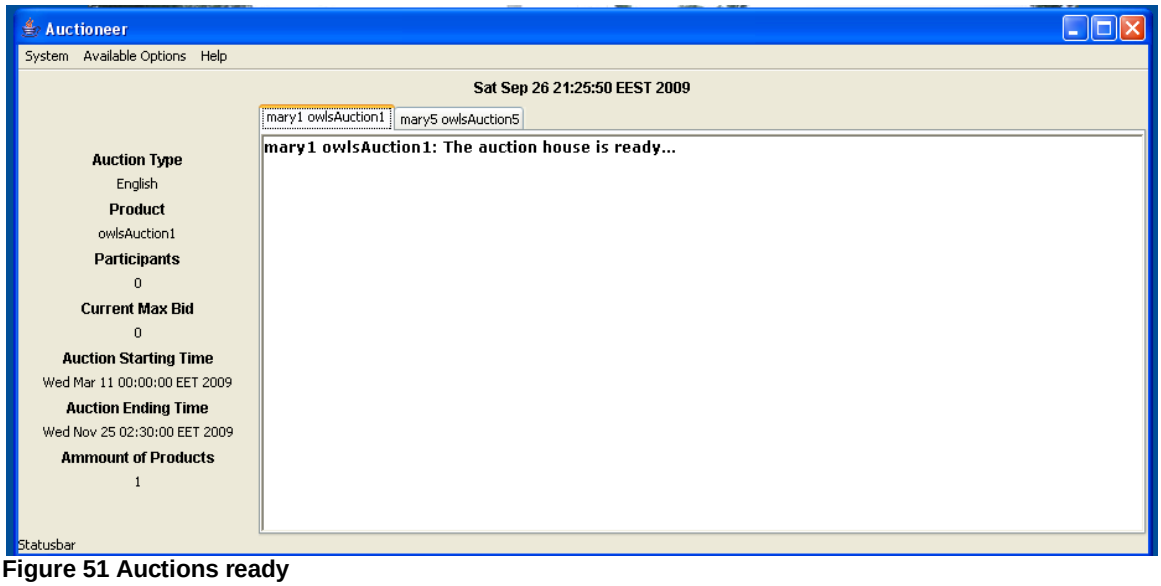


Figure 51 Auctions ready

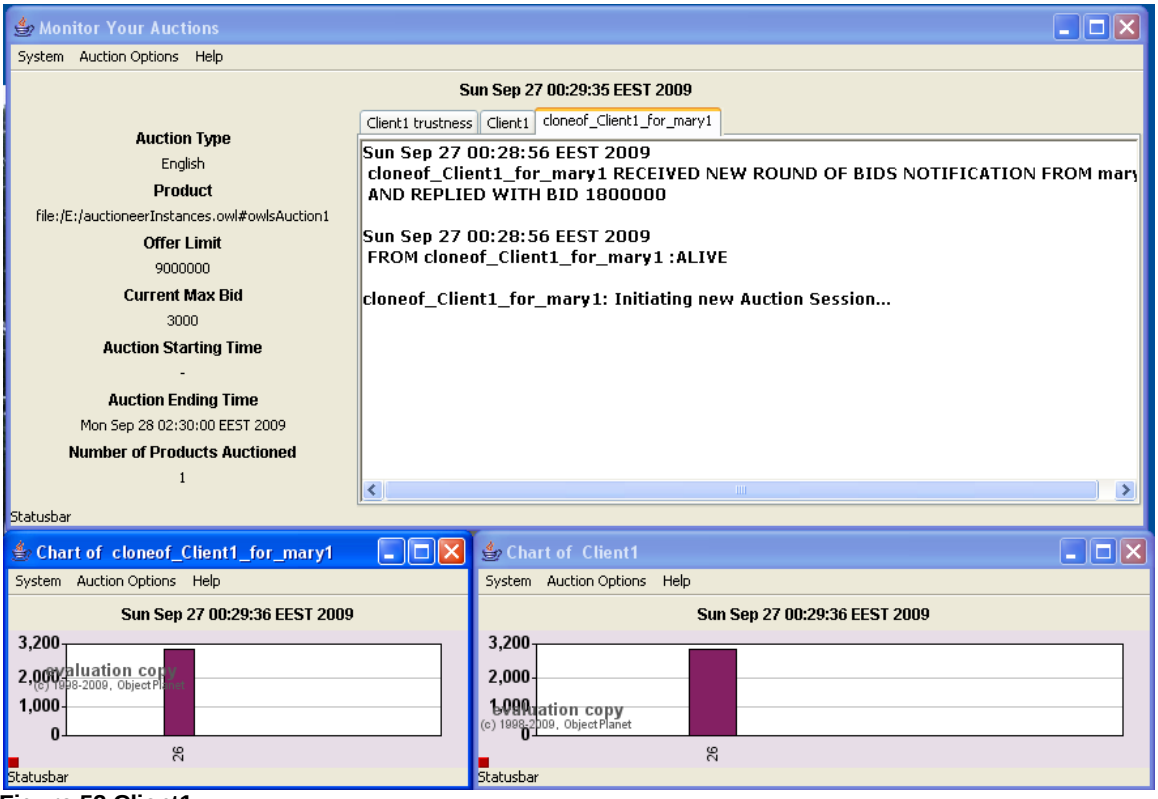


Figure 52 Client1

Then we initiate Client 1 (figure 43) that is going to take part only in the auction of the A-agent mary1

The same way we initiate Client 2 (figure 44) that is going to participate both in the auctions contacted by mary1 and mary2.

Client2 begin being active in mary1 as it is the auction that will end first but also monitor mary5 and so he finds out that mary 5 sells the item in a lower price so he starts monitoring both auctions waiting for the item sold in mary1 to become cheaper than the one sold in mary5.

We can see that from the moment the C-agent finds the cheaper auction mary5 until the time he informs CL-agent for mary1 to go to monitor mode the CL-agent sent 1 bid that is considered shilling bids by the C-agent. That is the reason an agent is though a shiller after he makes 2 shilling bids.

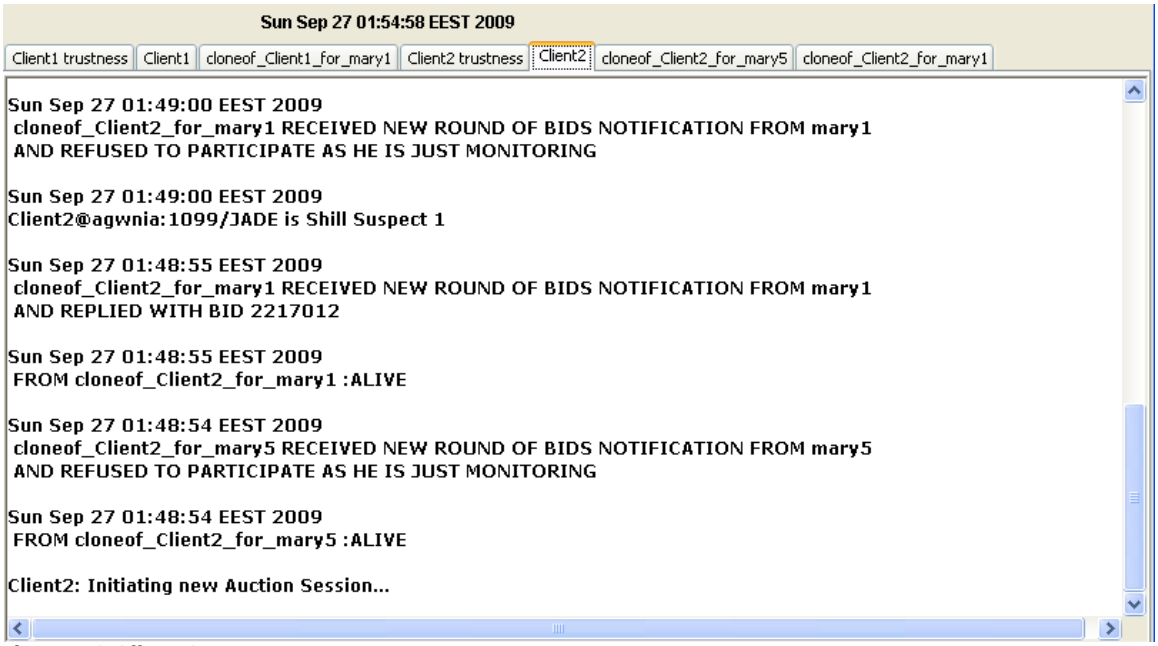


Figure 53 Client 2

And finally we initiate shiller who is an agent that participates both in mary1 and mary5 but his only purpose is to raise the price of the item sold in mary1.

As we can see in figure 45 the C-agent of client2 finds that the bids of the shiller agent are shilling bids. As we said he forgives the first one but when the second happens he is sure the shiller is trying to raise the price of the auction.

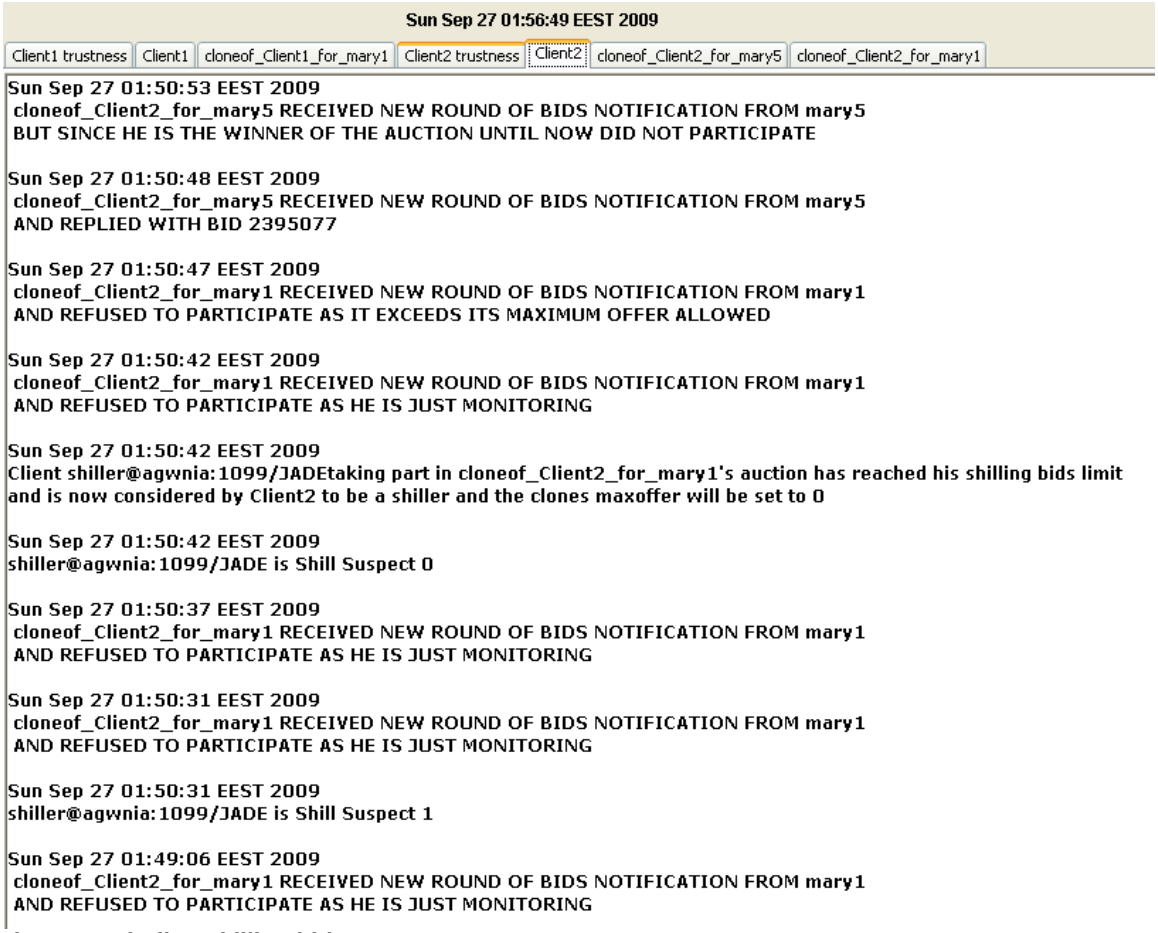


Figure 54 Finding shilling bids

When the shiller agent exceeds the number of shilling bids allowed Client2 stops taking part in the first auction and becomes active in the second auction.

9 Experimental Evaluation

To evaluate the performance of our agent using the bidding algorithm described in 7.4, we followed the authors' example and made an empirical evaluation. Here we defined three control models as a basis for comparison. These models simulate three plausible modes of behaviour in online auctions. In the first model (control C1), the agent randomly joins one auction and stays there until its reservation price has been reached or until the auction's end time is reached. The agent does not move to any other auction. Our second control agent deploys a greedy strategy (control C2). This agent picks the auction that has the closest end time where the current bid value for the item does not exceed the reservation price. If there is more than one possibility, it makes a random choice. The agent stays there until the reservation price has been reached or until the auction is over. If it has not purchased the item, it then moves to another auction using the same selection technique and repeats the process until its allocated time is over. In fact this is the model we follow in our work. In the last model (control C3), an agent picks an auction randomly from all the active auctions and bids in that auction. It stays there until its reservation price is reached or until the auction is over. If the agent is not successful, it randomly picks another auction and repeats the process until the allocated time is over or it successfully purchases the item.

We made 150 runs for the following experiments. These experiments were run in an environment with Tmax = 120 seconds, between 10 English auctions running concurrently, and for each auction, there are between 2 and 12 participants. So each time we initiated 4 C-agents from each model (C1, C2, and C3) and they randomly chose 4 of the 10 auctions to take part in. The C-agents of model C1 took part only in the first of the four auctions they chose, those of model C2 took part in all of them randomly and those of model C3 followed our strategy. The shilling detection ability was removed from the agents. We made sure that all the clients subscribe in the auctions they are interested in before the auction begins and so they are all there from its beginning. Also all the agents have the same max bid value (1000000). All auctions have 10 seconds difference in their starting time (Auction 1 begins at 0, auction2 at 10 auction3 at 20 etc). Although we have this starting time difference all agents have the same Tmax as each clone calculates as Tmax the difference between the auctions ending time and the time he got the first bid and since all agents are subscribed in their auctions before the beginning of the auction they will all receive the beginning bid at the same time when the auction starts.

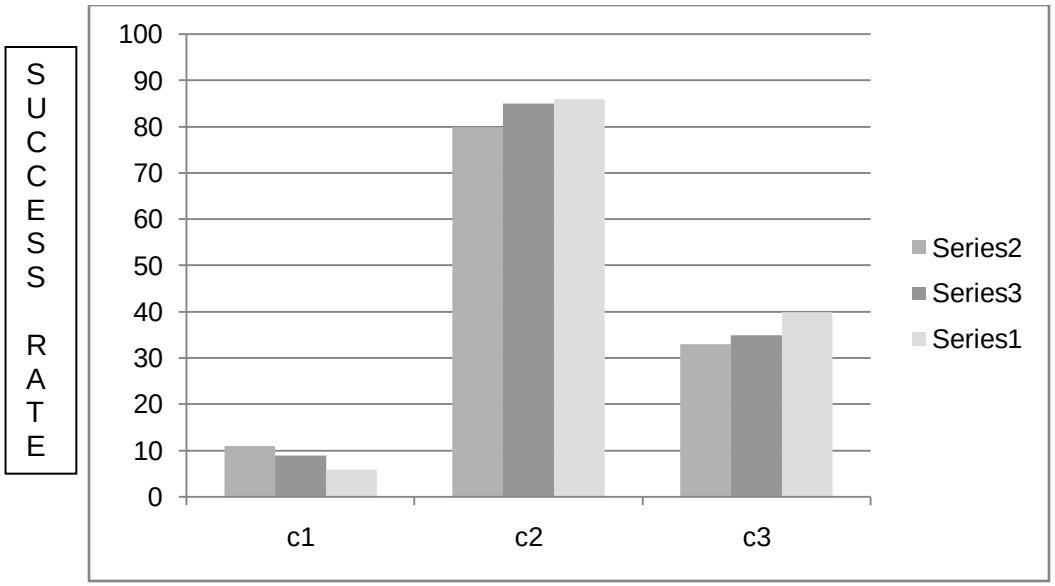


Figure 55 Experiment 1 Success rate results

In the first 50 runs β= 0.3 giving a low level desire for bargain behaviour. In runs 51-100 β= 2 making the bid value quickly reach max bid and in the final 50 β=50 making the bid value reach max bid even faster.

Figure 55 shows the performance of all the agents in terms of their success rate. The success rate is defined as the number of times, as a percentage, the agent is successful in obtaining the item. We run the experiment 150 times and divided the result into 3 groups as you can see in the diagram.

The performance of C2 is very good, since it takes every opportunity to bid in a terminating auction without considering other issues like payoff and time left to bid. C1 and C3 perform poorly due to their simplified behaviour of picking any auction randomly. C1 takes part only in one auction and so the chances of winning are much smaller than the chances of the other two models. C3 takes part randomly and so looses the chance to participate in all the auctions that end before his current selection.

It shows the agents final bid values (the bid values at which they acquired the item) as percentages of the average closing prices for all the auctions in the marketplace that closed within the period that the agents were active. The final bid value is considered as a very important factor in auctions analysis since it determines the closing price of a particular auction and the resulting payoff that the agent expects to get upon acquiring the item. This result leads us to conclude that our agents successfully acquire the item at a price lower than or equal to the reservation price, which is subsequently lower than the average closing price in all the auctions that closed within the period that they were active.

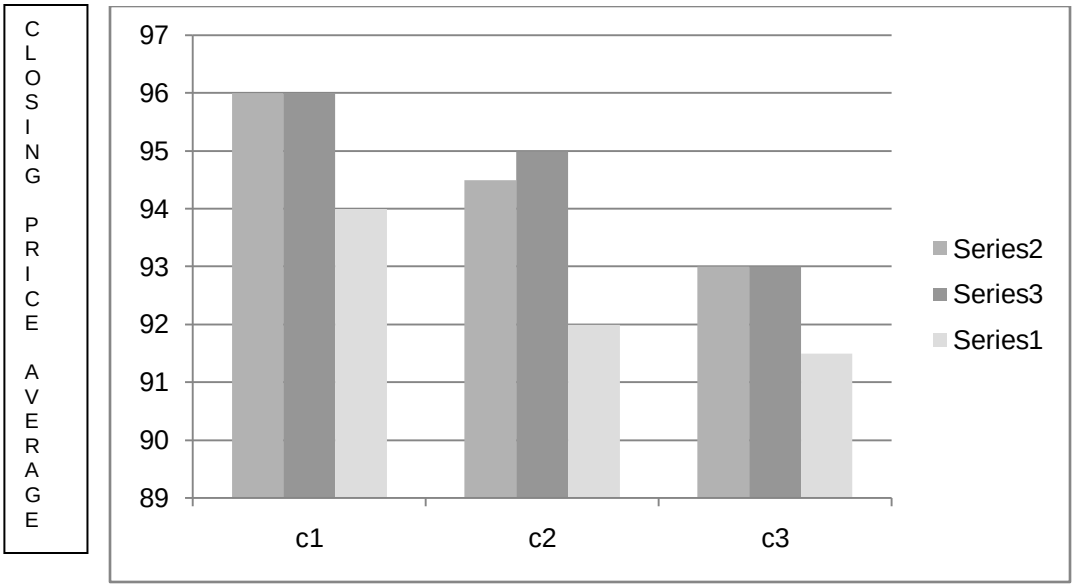


Figure 56 Closing Price Comparison

As we can see again from figure 56 in series 1 which shows the first 50 experiments when the agents are more conservative the closing prices average are lower whereas in the other two that are more aggressive they are higher.

Figure 57 shows the average payoff for each agent participating in the experiment. **Payoff** is calculated by subtracting the agent's bid value (when it successfully acquired the item) from the reservation price. The average payoff is the summation of all the payoffs divided by the number of times the agent successfully acquires the item. The performance of the three control models is poor because of their disregard for the payoff issue. Their main goal is to get the item at a price lower than or equal to the reservation price. As described in the experiments of [90] there are other beter compiations of the parameters of the algorithm that result in better payoff values.

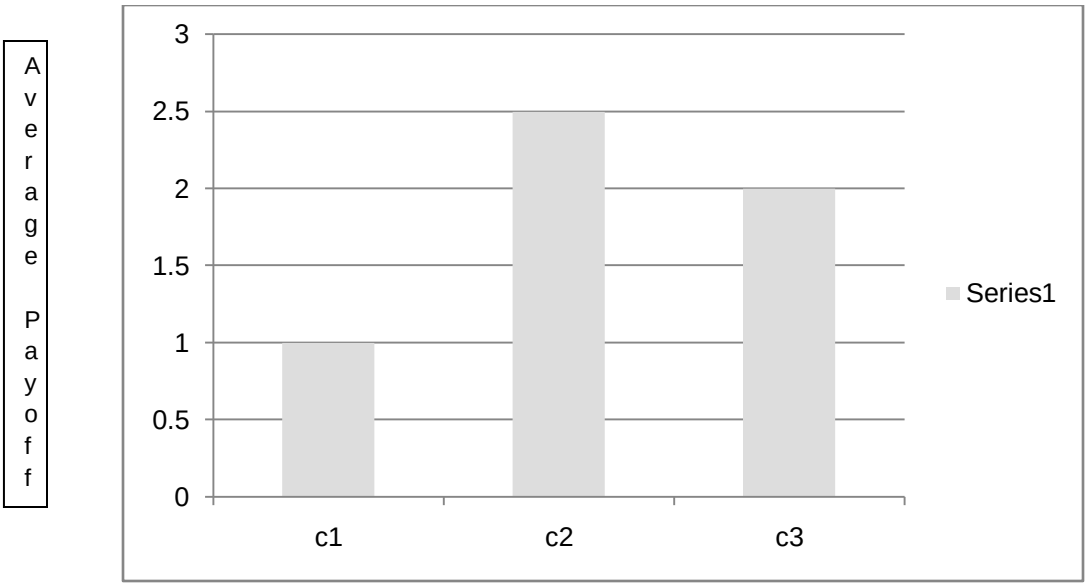


Figure 57 Average Payoff

Finally, Figure 58 shows the average time taken by the agents to acquire the item. The control models spend between *t = 40* and *t = 110*. This indicates that the control models try to take any opportunity to bid in a particular auction as soon as possible. We can see that in cases where the agent is more aggressive he wins in less time than in the cases he is conservative.

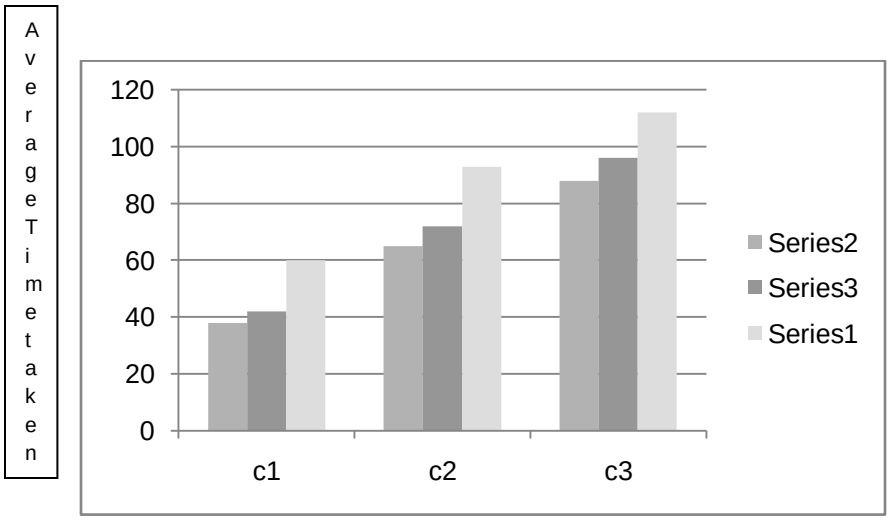


Figure 58 Average Time taken

To sum up we can see that our auction behavior has about the same behavior as the auctions in the experiments of [90] and in all the experiment's different views we presented method C2 which we used in our system appears to have the

better results. In [90] a variety of other cases of agents is tested and other better abilities of the bidding algorithm are described that our system cannot support directly but they can appear in it if the algorithm's parameters take the right prices. In fact further exploration and development of strategies for our bidding agent that make him show the behaviors the other agents in experiments of [90] show could be part of our future work.

10 Conclusions and future work

We have presented the design and implementation of a system that integrates three emerging technologies; intelligent software agents, peer-to-peer networking and the Semantic Web. It is an architecture for an agent-based virtual marketplace and is intended as a platform for research in multi-agent systems, ontologies, peer-to-peer networks and automatic negotiation, as well as an application for experimentation with these technologies. One of our motivations in creating this platform has been to demonstrate the power of combining the three technologies in facilitating the participation of human users in next-generation e-Commerce transactions. The system realizes a number of auction scenarios as a general negotiation framework among users, while maintaining a flexible design that allows it to be easily extended with new scenarios and techniques.

Our project deals with many issues concerning most of e-Commerce phases and proposes novel and feasible solutions. Semantic languages and formalisms are being used for conceptualizing negotiation protocols and providing product domain characterization. They are also used for semantically publishing and discovering available resources on the network. Local ontologies capture concepts and relations of products and allow retailers to add metadata to their descriptions, following a common schema definition, while customers analyze those ontologies and find the matching result between their preferences and the items offered using a reactive rule that is dynamically created according to the user's preferences. The agents also gather information about the A-agents and from that information with the use of reactive rules they take trust decisions that affect their behavior towards the A-agents.

Moreover during the progress of an auction the C-agents have the ability to detect agents that attempt to do shilling. It is important to point out that both setting the trust level and detecting the shilling agent is done form the clients part and the user does not have to base his actions and trust to information the auction house gives to him but acts individually and independently from the A-agent.

Software agent skills are exploited to automate human user's participation in multiple, possibly interrelated, auctions and to provide advanced means of e-Commerce transactions. The multi-agent architecture is flexible enough to allow experimentation with various negotiation protocols, bidding algorithms and job allocation techniques. We have shown how our agents segment the job of

participating into multiple auctions into subtasks and cooperate appropriately to accomplish them in a manner consistent with their owner's preferences. Intelligent software agents perform various functions, such as auction bidding and monitoring, statistical analysis of auction data from the C-agent, background execution and more.

We have also described the peer-to-peer infrastructure that is utilized to provide the medium for remote users to contact and negotiate, while at the same time distributes computational and knowledge resources. The architecture of our peer-to-peer network is scalable, permitting various alternative models to be implemented, enhancing its capabilities.

In addition, two auction protocols have been implemented that cover many issues of multi-party negotiation; the English auction, which is a multi-round open model and the Vickrey auction, which is single-round and sealed-bid. We have shown that extending the platform with new negotiation protocols is a non-trivial task, affecting only specific components of the system.

The user interface was implemented in such a way that it can dynamically adjust to any domain ontology the user may use as all the pop ups concerning the domain ontology are not static but are dynamically designed with the use of the OWL-API. So if the user decided to use a different or more complicated domain ontology this could not be a problem. The same happens with the JESS rules as with the use of the Ontology Bean Generator the JavaBeans of any new ontology used will be created automatically and the user would have to make some little changes in the function that calculates the matching result.

The users can also see charts that represent the progress of their auctions dynamically. In the charts we can see the progress of each auction separately, the average price of all the auctions one participates in, who made each bid in an auction and other statistical results.

The auctions strategy during the negotiation process is affected by many parameters; the information he gathers from other agents he trusts and the level of trust he decides for each A-agent, the matching result of the clients preferences and the items characteristics, the maximum bid he is willing to give in each auction, the number of auctions he takes part in and the remaining time in each auction. All these parameters affect how aggressive or conservative the clone will be and so clones of the same agent can show different behaviours.

The agents' communication is contacted following the FIPA specifications and with the use of FIPA-ACL messages. This gives our agents the ability to be able to communicate with agents others have created that also follow the same specifications and use ACL messages. We can see this in our platform as the shilling agent we created follows a completely different logic but manages to communicate with our agents and takes part in auctions.

Our algorithms implementation quarantines that we will always come to an end. First of all the user knows that the auctions he chose will be taken as target auctions one after the other according to their ending time beginning from the one that ends first and finishing with the one that ends last. It is also clear that only one CL-agent is active each time and the rest are just monitoring until their turn comes so there is no chance of winning in more than one auctions. Inside each auction negotiation from which the C-agent will pass it is assured that the negotiation will begin and end whether the CL-agent wins or loses as we follow the FIPA specifications for the implementation of the auctions' (English, Vickrley) and so all the cases are taken under consideration. The Design of the agent negotiation also assures that the CL-agent and the A-agent will never lose contact as they negotiate in the same room and so there is no way for those two to lose contact. The only problem that could happen is the CL-agent to lose contact with the C-agent that crated it but we consider that all agents are moved to host that will always be available and so the agents will not lose contact but even if they lose the CL-agent is completely independent in his negotiation logic and will vontinue until the contact with his C-agent is restored. So the algorithm starts with the beginning of the first auction and ends either when an auction is won or with the end of the last auction the user selected.

Nevertheless, the approach we have presented is open to further improvement and modifications.

As we described each agent auction is held in a room, which the user must join to become eligible to participate and bid in the auction. Certain authorization mechanisms could be utilized to control the process of joining. As a result, various alterations of auction sessions can be generated by this pattern; private auctions or secure auction sessions could be implemented by extending the authorization process for a peer to join the corresponding peer group.

Another interesting suggestion is to create a graphical interface where a user could place his own preferences about the level of trust he wants to have towards another agent and according to these preferences the rules that decide the level of trust would be created. But this is a rather complicated work because the user could easily make mistakes and create rules that retract each other and this could create an endless circle of retractions. So the system must have the ability to find such cases and warn the user.

More complicated negotiation algorithms can be implemented and easily adapted from our system. And this way the clients could have the ability from various algorithms to select which one they want to follow. But our main line of work is to further explore the development of strategies for our bidding agent. Since the environment in which the multiple auctions are running is highly dynamic, we could extend this work by using an evolutionary approach based on genetic algorithms (GAs). GAs will be used to determine the relative success of these strategies and how these strategies can evolve over time to better fit their environment. The performance of the agent is very much influenced by the strategy employed which, in turn, relates to the values of k and β *in* the given

tactics and the weights for each tactic when combined. Different strategies may perform well in some environments but may perform poorly in another. The numbers of strategies that can be employed are endless and the search space is huge. To address this issue, we intend to use GAs to search for the most successful strategies in predefined environments in an off-line fashion. This knowledge can then be codified into an agent's online reasoning behaviour so that it can determine the best strategy to employ in the prevailing circumstances.

Finally, an important aspect of our future objectives is to enhance the graphical user interface with graphics and diagrams not only for following the progress of auctions and for viewing the statistical information gathered by C-agents but showing more complicated results and metrics that can be mathematically and statistically calculated from the information CL-agents and C-agents gather.

Bibliography

[1].Theodore Patkos, Dimitris Plexousakis: A Semantic Marketplace of Peers Hosting Negotiating Intelligent Agents. CAiSE Short Paper Proceedings 2005

[2]. Charles L. Forgy, Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem", Artificial Intelligence 19, 17-37, 1982.

[3]. Theodore Patkos, Dimitris Plexousakis: A Semantic Marketplace of Negotiating Agents. AP2PC 2005: 94-105

[4]. Holger Knublauch, “An AI tool for the real world Knowledge modeling with Protégé”, JavaWorld.com, June 2003.

[5]. Setchi R. and Lagos N., “Ontology Development And Merging Using Protégé”, Cardiff School of Engineering, Cardiff University, UK, 2003.

[6]. V. Shmatikov and C. Talcott, “Reputation-Based Trust Management,” *Journal of Computer Security*, Special Issue on Selected Papers of WITS 2003

[7]. A.A. Selcuk, E. Uzun, and M.R. Pariente, “A Reputation-Based Trust Management System for P2P Networks,” In *Proceedings of the 4th IEEE/ACM International Symposium on Cluster Computing and the Grid* (CCGrid 2004), Chicago, USA, April 2004

[8]. S. Ganeriwal and M.B. Srivastava, “Reputation-Based Framework for High Integrity Sensor Networks,” In *Proceedings of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks* (SASN 2004), Washington DC, USA, pp. 66-77.

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

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

[11]. 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.

[12]. S. J. Brams and A. D. Taylor. *Fair Division: From Cake-cutting to Dispute Resolution*. Cambridge University Press, 1996.

[13]. Interagency Working Group on Information Technology Research and Development, *Grand Challenges: Science, Engineering, and Societal Advances Requiring Networking and Information Technology Research and Development*,

National Coordination Office for Information Technology Research and Development,USA,2003.http://www.nitrd.gov/pubs/200311_grand_challenges.pdf

[14]. IST Advisory Group, *Software technologies, embedded systems and distributed systems: A European strategy towards an Ambient Intelligence environment*, European Commission, 2002.

[15]. I. Foster and C. Kesselman (Eds.), *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 2004.

[16]. I. Foster, N. R. Jennings and C. Kesselman, Brain meets brawn: Why Grid and agents need each other, in *Proceedings of the Third International Conference on Autonomous Agentsand Multi-Agent Systems*, 8–15, ACM Press, 2004.

[17]. The Semantic Web - Scientific American May 17, 2001 Tim Berners-Lee, James Hendler and Ora Lassila .A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. http://www.sciam.com/article.cfm?SID=mail&articleID=00048144-10D2-1C70-84A9809EC588EF21.

[18]. D. S. Milojicic, V. Kalogeraki, R. Lukose, Rajan, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins and Z. Xu, *Peer-to-Peer Computing*, HP Technical report HPL-2002-57, 2002.

[19]. U. Endriss and N. Maudet. Welfare engineering in multiagent systems, in A. Omicini, P. Petta, and J. Pitt, editors, Engineering Societies in the Agents World IV, Lecture Notes in Artificial Intelligence 3071, 93–106, Springer, 2004.

[20]. M. Lemaître, G. Verfaillie, H. Fargier, J. Lang, N. Bataille, and J.-M. Lachiver. Equitable allocation of earth observing satellites resources, in *Proceedings of the 5th ONERA-DLR Aerospace Symposium*, 2003.

[21]. P. Cramton, Y. Shoham, and R. Steinberg, editors. *Combinatorial Auctions*. MIT Press, 2006.

[22]. Y. Chevaleyre, U. Endriss, J. Lang, and N. Maudet. Negotiating over small bundles of resources. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*. ACM Press, 2005.

[23]. Y. Fujishima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 548–553, Morgan Kaufmann Publishers, 1999.

[24]. A. Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, March 2001.

[25]. J. F. Kurose and K. W. Ross, *Computer Networking (Second Edition)*. Addison-Wesley, 2002.

[26]. R. H. Bordini, M. Dastani, J. Dix, A. El Fallah Seghrouchni (Eds.), *Multi-Agent Programming: Languages, Platforms and Applications*, Springer, 2005.

[27]. A. Kuflik, Computers in control: rational transfer of authority or irresponsible abdication of autonomy, *Ethics and Information Technology*, 1(3): 173–184, 1999.

[28]. 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.

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

[30]. 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/

[31. 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/.

[32]. Grigoris Antoniou, Frank van Harmelen (2004). “A Semantic Web Primer”. MIT Press 2004.

[33]. JADE http://jade.tilab.com/

[34].JESS http://herzberg.ca.sandia.gov/

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

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

[38]. P.A. Bonatti, C. Duma, D. Olmedilla, *et. al.*, “AnIntegration of Reputation-Based and Policy-Based TrustManagement,” In *Proceedings of the Semantic Web Policy Workshop*, Galway, Ireland, November 2005.

[39]. Y. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss, “REFEREE: Trust Management for Web Applications,” *Computer Networks and ISDN Systems*, Vol. 29, 1997, pp. 953-964.

[40]. Trust Management for Public-Key Infrastructures," In *Proceedings of the Security Protocols: 6th International Workshop*, Cambridge, UK, April 1998, LNCS 1550, Springer-Verlag, 1998, pp. 59-63.

[41]. S. Ruohomaa and L. Kutvonen, "Trust Management Survey," In *Proceedings of the iTrust 3rd International Conference on Trust Management*, May 23-26, 2005, Rocquencourt, France, Springer-Verlag, LNCS 3477, May 2005, pp.77-92.

[42]. R. Patel, H. Xu, and A. Goel, "Real-Time Trust Management in Agent Based Online Auction Systems," In *Proceedings of the 19th International Conference on*

Software Engineering and Knowledge Engineering (SEKE'07), Boston, USA, July 2007, pp. 244-250.

[43] Alfarez Abdul-Rahman and Stephen Hailes. Supporting Trust in Virtual Communities. *Proceedings of the 33rd Hawaii International Conference on System Sciences*, Maui, Hawaii, January 2000.

[44] L. Mui, M. Mohtashemi, C. Ang, P. Szolovits, and A. Halberstadt. Ratings in Distributed Systems: A Bayesian Approach. *Proceedings of the 11th Workshop on Information Technologies and Systems,* New Orleans, LA, December 2001

[45] Lik Mui, Mojdeh Mohtashemi, and Ari Halberstadt. A Computational Model of Trust and Reputation. *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, Maui, Hawaii. January, 2002.

[46] S. Marsh. Formalizing Trust as a Computational Concept. PhD Thesis, Department of Computing Science and Mathematics, University of Sterling, April 1994.

[47] Java website, http://www.java.sun.com

[48] Wooldridge M., Jennings N.: Reasoning about rational agents. MIT Press, 2000.

[49] The Protégé OWL API, http://protege.stanford.edu/plugins/owl/api/ .

[50] W. Vickrey, Counter speculation, auctions, and competitive sealed tenders, Journal of Finance 16 (1) (1961) 8–37.

[51] M. Osborne, R. Rubistein, A Course in Game Theory, MIT Press, 1994.

[52] E. Rasmussen, Games and Information, Blackwell Oxford, 1994, second Edition.

[53] P. Klemperer, Auction theory : A guide to the literature, Journal of Economic

Surveys 13 (3).

[54] Federal communications commission : All about auctions, FCC Report (September 1999).

[55] eBay, http://www.ebay.com/.

[56] Moai, http://www.moai.com/.

[57] Agorics Inc., Going Going Gone! A Survey of Auction Types, http://www.agorics.com/new.html.

[58] Market Design Inc., http://www.agorics.com/new.html.

[59] P. Milgrom, Auction Theory for Privatization, Cambridge University Press, 1998.

[60] P. Wurman, Market structure and multidimensional auction design for computational economies, Ph.D. thesis, University of Michigan (1999).

[61] J. Rosenschein, G. Zlotkin, Rules of Encounter : Designing Conventions for Automated Negotiation Among Computers, Artificial Intelligence, MIT Press, 1994.

[62] T. Sandholm, Limitations of the Vickery auction in computational multiagent systems, in: Proceedings of the Second International Conference on Multiagent Systems (ICMAS-96), Kyoto, Japan, 1996.

[63] Forrester Research, Dynamic trade research brief,

http://www.forrester.com/ER/Research/Brief/0,1317,1481,FF.html.

[64] S. Klein, Introduction to electronic auctions, International Journal of Electronic Markets 7 (4).

[65] E. Turban, Auctions and bidding on the Internet : An assessement, International Journal of Electronic Markets 7 (4).

[66] M. Benyoucef, K. Keller, S. Lamoureux, J. Robert, Towards a generic enegotiation platform, in: Sixth Int. Conference on Re-Technologies for Information Systems, Austrian Computer Society, Zurich, Switzerland, 2000, pp. 95–109.

[67] P. Maes, R. Guttman, A. Moukas, Agents that buy and sell: Transforming commerce as we know it, Communications of the ACM 42 (3).

[68] G. Kersten, S. Noronha, J. Teich, Are all e-commerce negotiations auctions?, in: COOP’2000 : 4th International Conference on the Design of Cooperative Systems, Sophia-Antipolis, France, 2000.

[69] A. Chavez, P. Maes, KABASH: An agent marketplace for buying and selling

goods, in: 1st Int. Conference on Electronic Commerce, IECE’98, Soeul, Korea,

1998.

[70] D. Parker, L. Ungar, D. Forster, Agent Mediated Electronic Commerce, Springer, 1999, Ch. Accounting for Cognitive Costs in On-line Auction Design, pp. 25–40.

[71] J. Hu, D. Reeves, H. Wong, Agent service for online auctions, in: Proceedings of the AAAI-99Workshop on AI for Electronic Commerce, AAAI Press, 1999.

[72] M. Benyoucef, K. Keller, An evaluation of formalisms for negotiations in ecommerce, in: P. Kropf, et alt. (Eds.), Distributed Communities on the WEB, Vol. 1830 of LNCS, Springer, 2000, pp. 45–54.

[73] D. Ferguson, C. Nikolaou, J. Sairamesh, Y. Yemini, Market Based Control of

Distributed Systems, scott clearwater Edition, World Scientific Press, 1996, Ch. Economic Models for Allocating Resources in Computer Systems.

[74] OMG, Negotiation facility final revised submission, Technical report, http://www.oms.net (Mar. 1999).

[75] FIPA Specifications, http://www.fipa.org/specifications/

[76] BARTOLINI, Claudio; PREIST, Chris; JENNINGS, Nicholas R.. A

Software Framework for Automated Negotiation. United States: Springer-verlag, 2004. 3390v.

[77] TAMMA, V. et al. Ontologies for supporting negotiation in e-commerce. Engineering Applications Of Artificial Intelligence, Liverpool, p. 223-236. 05 nov. 2005.

[78] FARATIN, P.; SIERRA, C.; JENNINGS, N. R.. Using Similarity Criteria to Make Issue Trade-offs in Automated Negotiations. Artificial Intelligence, Cambridge, p. 205-237. 01 dez. 2002.

[79] Anthony, P., Hall, W., Dang, V. and Jennings, N. R. (2004) Autonomous Agents for Participating in Multiple On-line Auctions. In: *IJCAI Workshop on E-Business and the Intelligent Web*, Seattle, USA.

[80] Bădică, C. Batita, A. (2006) Implementing rule-based mechanisms for agent-based price negotiations , in 2006 ACM symposium on Applied computing

[81] Kevin M. Lochner, Michael P. Wellman, "Rule-Based Specification of Auction Mechanisms," aamas, vol. 2, pp.818-825, Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2 (AAMAS'04), 2005

[82] Wolfgang Nejdl, Daniel Olmedilla, Marianne Winslett PeerTrust: Automated Trust Negotiation for Peers on the Semantic Web Workshop on Secure Data Management in a Connected World (SDM'05)

[83] Vinicius da S. Almendra, Daniel Schwabe Trust Policies for Semantic Web Repositories In proceedings of 2nd International *Semantic Web Policy* Workshop (SWPW'06)

[84] Haiping Xu, Sol M. Shatz, Christopher K. Bates A Framework for Agent-Based Trust Management in Online Auctions Proceedings of the Fifth International Conference on Information Technology: New Generations 2008

[85]A. A. Selcuk, E. Uzun, M. R. Pariente A Reputation-Based Trust Management System for P2P Networks Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid Pages: 251 - 258

[86] Golbeck, J., Parsia, B., Hendler, J. Trust Networks on the Semantic Web. ISWC'04.

[87] http://jade.tilab.com/community-3rdpartysw.htm

[88] http://www.swi.psy.uva.nl/usr/aart/beangenerator/index25.html

[89] G. Weiss, editor. Multiagent systems: A modern approach to distributed artificial intelligence. MIT Press, 1999.

[90] Anthony, P., Dang, V. D., Hall, W., and Jennings, N. R. 2001. Autonomous agents for participating in multiple online auctions. In Proceedings of the IJCAI Workshop on E-Business and Intelligent Web. 54--64.