Using Ontologies to Structure Ontology-Centric Multi-Agent Systems (OCMAS)

By

Muhammad Jan Khan

2011-NUST-MS Phd-IT-30

Supervisor

Dr. Peter Bloodsworth

Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree of Masters in Information Technology (MSIT)

In

School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(June 2015)

Approval

It is certified that the contents and form of the thesis entitled "*Using Ontologies to Structure Ontology-centric Multi-Agent System*" submitted by **Muhammad Jan Khan** has been found satisfactory for the requirement of the degree.

Advisor: Dr. Peter Bloodsworth

Signature: ________________
Date: ________________

Committee Member 1: Dr. Sohail Iqbal____

Signature: ________________
Date: ________________

Committee Member 2: Dr. Sarah Shafiq

Signature: ________________
Date: ________________

Committee Member 3: Mr. Shamyl Bin Mansoor

Signature: ________________
Date: ________________

I dedicate this work to my Parents for their affection and support.

Certificate of Originality

It is declared that the research work titled "*Using Ontologies to Structure Ontology-Centric Multi-Agent System*" is my own work to the best of my knowledge. It contains no materials previously written or published by any other person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST or any other educational institute, except where due acknowledgment, is made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the projects design and conception or in style, presentation and linguistic is acknowledged. I also verified the originality of contents through plagiarism software.

Author Name: MUHAMMAD JAN KHAN

Signature: ____________________

Acknowledgment

I am truly thankful to the SEECS faculty for their direction, cooperation and guidance all through my research work. I want to express my profound appreciation to my postulation research supervisor **Dr. Peter Bloodsworth** (NUST) for his bolster, vision, mentoring and direction. I benefited from his valuable help, input and critique. I am likewise grateful to my advisory group individuals **Dr. Sohail Iqbal**, **Dr. Sarah Shafiq** and **Mr. Shamyl Bin Mansoor** for their availability, support, and helpful input during my research.

I extend my appreciation to my schoolmate and companions at SEECS (NUST) for sharing input and learning. Specifically, I wish to say thanks to **Sehrish Amjad** and **Rana Faisal Munir** for their encouragement throughout and assisting me in the concept understandings of framework design. Lastly, I am thankful to my family for their bolster and love.

TABLE OF CONTENTS

List of Abbreviations

Abbreviations	Descriptions
MAS	Multi-Agent System
JIAC	Java Intelligent Agent Component ware
JADE	Java Agent Development Environment
OWL	Ontology Web Language
RDF	Resource Description Framework

Abstract

Designing a multi-agent system requires complete knowledge regarding the domain. A methodology is required that allows us to adapt to the changes in the system without restructuring it. Due to ever changing system requirements the behavior of the agents may need to change frequently. Such changes often require costly system redesigning and coding. Therefore a framework (OCMAS) is proposed to define the behavior of agents in a frame-based ontology where agents are created with a set number of abilities. The behavior or manner of exactly how they do their job is defined in the ontology. When initialized an agent looks up into the ontology which defines exactly how the behavior is executed. However, the actual realization of an agent's behavior is implemented in a rule-engine that is configured from the ontology. We described a group of agent abilities and collaboration mechanisms in an ontology and then tested it by altering the behavior through editing the ontology. Two prototype multi-agent auction systems have been developed as proof of concept. One uses the conventional JADE approach and the second uses the proposed model which allows the creation of reconfigurable agent behaviors at real-time from one auction protocol to another. The evaluation showed that system has the potential to successfully change the agent's behavior. In terms of system performance, the results indicated that with sufficient resources in terms of memory and CPU, the system works reasonably efficiently.

Chapter 1

Introduction and Motivation

1.1 INTRODUCTION

In multi-agent systems, numerous agents interact and cooperate to do a particular arrangement of goals. The coordination and collaboration between agents having different knowledge and problem solving abilities normally makes conceivable the accomplishment of global objectives that cannot be generally accomplished by a solitary agent working in isolation. Along these lines multi-agent systems are considered as a response to various weaknesses of general problem solving impediments which primarily include deficient knowledge requirements, and restricted computational assets. Multi-agent systems are especially valuable in the development of open, dynamic and adaptive systems.

However, designing a multi-agent system requires complete domain knowledge. A comprehensive methodology is required that allows us to adapt to changes in the system without restructuring it. A lot of methods have been devised for designing and developing multi-agent systems, but very few of them address the structure of agent's behavior which is an essential part in designing a good system implementation. Due to ever changing system requirements the behavior of the agents may need to change frequently. Such changes often require costly system redesigning and coding. Therefore an idea is proposed to define the behavior of agents in a frame based ontology where agents are created with a set number of abilities, but the behavior of exactly how they do their job is defined in the ontology. Information from the ontology is used to define how an agent performs its actions. This ensures that agents perform as is specified in the ontology thereby achieving the overall objectives of the system. Whenever a change is needed in the system, only the ontology needs to be changed and agents synchronize with the updated ontology. By using the ontologies to structure agents' behavior, each agent will share a common understanding in the domain of interest. The design of a multi - agent system can be generic, and changes to the behavior of agents can be changed with modifications in the ontology, while the rest of the system remains intact. Agents of one system can also communicate with the agents of another system using the protocols defined in the ontology.

Due to the advancements in the field of communication technology, mobile phone devices have become very common and each mobile phone can work as an agent due to the available processing capabilities. So to devise a multi-agent system with those millions of mobile agents around us, the designing task is quite tough. Because, requirements of such systems may change dramatically and it is barely possible to design the system to adjust at real-time with such changing needs. So ontology centric multi-agent system will provide the flexibility in design where the agents are defined with a role to perform, but their behavior is defined in the frame based ontology.

In the proposed methodology the ontology layer is prepared whilst the architecture is being defined and it is situated at the core of the system. Every agent is intended to fulfill a particular task in the system; however it is planned in a nonspecific way. Ontology layer characterizes the rules, heuristics and measurable attributes that recognize the domain specific parts of agents. Every agent in the system uses this ontology layer along with the instances of the ontologies to produce a knowledge base which helps in specifying its behavior. To perform this, agents should be made with the ability to make utilization of knowledge base at run-time to construct their behaviors. By presenting an abstraction level between the coding and agents' behavior, brings about making more valuable editable systems. Modifications and altering of the ontology layer and their related instances does not oblige any significant changes in the coding of the individual agent of the system and it significantly underpins reusability, portability in context of the problem domain.

1.2 MOTIVATION

Headways in the field, has made the use of electronic gadgets very common and in reach of a number of users. A large number of vendors are now proving support to use these electronic gadgets for their systems and software. This has drastically increased the number of users for such software. Since software tend to evolve rapidly and not many systems designs support the system to adopt these evolutionary changes at real-time, some methodology is required to satisfy the said task. Ontology driven multi-agent framework will give the adaptability in design where the agents are characterized with a role to perform, however their behavior is characterized in the frame-based ontology.

The motivation of this research work is to show the utilization of ontologies to define agents' behavior in a multi-agent system. Since the use of semantic web in multi-agent frameworks is generally a new idea, very few industry applications have been created around this model. Thus, planning an architecture utilizing ontologies for multi-agent systems is a testing errand yet intriguing experience for my thesis research.

1.3 PROBLEM STATEMENT

When a multi-agent system is being developed, it not only brings with it the conventional issues attached to the distributed computing but also the individual issues of the agent's behavior. Such as when the individual behavior of the agents is combined, it gives totally different results as a whole than a system was intended to do. Due to this reason the system developers need to adopt a better approach to engineer the system while avoiding the above mentioned issues. A comprehensive methodology to deal with the ever changing behavior of agents is required that facilitate changes in the agent's behavior without restructuring / re-coding the system.

Another way is the use of ontologies to structure the agent behavior. Agents still got some roles to perform, but exactly how they behave is defined in the ontology. By using ontologies, each agent will share a common understanding in the domain of interest. This approach will bring many advantages to the system: the design of the multi-agent system can be generic and changes in behavior of the agents can be changed by modifications in the ontology and the rest of the system remains intact. One important question here is why and when a system requires change in its functionality/behaviors.

Changes in software of some organization happen usually because of one of four reasons, which can possibly overlap.

- Operational-Changes influence the way the continuous operations of the organization are performed, for example, the automation of a specific department.
- Strategic-Changes happen in the vital business objectives, e.g., changing business focus from an in-patient to an out-patient in a medical domain.
- Cultural-Changes influence the fundamental rationalities in an organization by which the business is performed, e.g., employing a constant quality improvement (CQI) mechanism.
- Political-Changes in staffing happen essentially for political reasons of different sorts, for example, those that happen at top support employment levels in governments.

1.4 RESEARCH METHODOLOGY

The main objective of this research work is to describe in detail OSMAS architecture, testing the designed framework by developing a prototype system and then evaluating the results in comparison to the current state of the art framework available.

The methodology involved defining the problem of real-time reconfiguration of multi-agent systems, followed by extensive literature review of current MAS architectures and reconfiguration methods. Furthermore, the designed framework for ontology-centric multi-agent systems (OCMAS) is proposed, and a prototype system is built on that framework. Also, another prototype system is developed using state of the art JADE framework, and both the systems are compared using a number of matrices. The designed OCMAS based system is developed using JAVA, JADE, Protégé, JENA, and DROOLS as a proof of concept. Lastly, to check the reliability and efficacy of the proposed architecture and to answer the research questions, the system is tested by making changes in the ontologies and monitoring the behavioral changes of agents in auction management multi-agent system.

1.5 RESEARCH HYPOTHESIS

The hypothesis of this work is broken down into a set of research questions which are individually answered and at the end of the work allows evaluating the hypothesis. The evaluated results are contrasted with the expected results which are assessed by taking after the proposed assessment methodology over the proposed system. The evaluation methodology utilized here to check the accuracy of this research includes a bartering/auction system as case study whose runtime information has been utilized to infer the actual results.

Research Hypothesis: -Using an ontology centric approach to define the behavior of an agent is more adaptable in term of reconfiguration?

Adaptability is the capacity of the domain specific reconfigurable model to adjust at real-time and scale itself to grow and contract at the runtime with respect to the evolving needs. The above clause evaluates that the proposed model ought to give adaptability/reconfiguration in the changing requirements and constant refinement in the system at runtime when contrasted with the already available models. Here reconfiguration is the capacity of the proposed OCMAS model to include, delete and modify the agent behaviors.

RESEARCH QUESTIONS

Q1: Why frame-based ontologies are appropriate for structuring agent's behavior?

Q2: How to design agents to reference and use ontologies?

Q3: How to handle the overhead that ontology-centric approach may impose on systems and to ensure that system performance is not affected?

Q4: How a developed multi-agent system can be tested and evaluated with changing behavior?

1.6 DOCUMENT ORGANIZATION

This document is organized into the following chapters:

Chapter 2 presents writing, study of the state of the art approaches for developing adaptive multi-agent systems.

Chapter 3 presents the design and architecture of the OCMAS framework. It additionally presents the details of each and every component of the OCMAS framework along with the implementation details with the help of a prototype.

Chapter 4 shows the evaluation details of the OCMAS framework.

Chapter 5 presents conclusions and mentions some future research options in the area.

Chapter 2

Literature Review

“Because ideas have to be original only with regard to their adaptation to the problem at hand, I am always extremely interested in how others have used them”
Thomas Edison

This chapter presents the review about the background study related to the multi-agents development frameworks and real-time configuration of software in order to make it adaptable to the changes. Agent based systems are quite different from the conventional systems because agents are active and the conventional systems are passive. So agents in a multi-agent system are built with the objective of having re-configurable behaviors. Despite the availability of a number of techniques and tools for agent based software development, there is still the need of some unified and mature way. Current state of the art agent platforms requires system developers to structure the behavior of agents in predefined methods. Also, this structure varies from one agent platform to another, i.e. coding the agent behavior in JIAC platform is altogether different from coding behavior in JADE platform. The difference in coding approaches causes the agent based system’s maintenance more costly. This maintenance cost would further be increased if the agent based systems are to be built where agent’s behavior is continuously updated with changing requirements.

2.1 RELATED WORK

Let’s first explain briefly what multi-agent systems are and what is the existing state of the art in agent based systems development. We then explain the state of the art approaches that assist in the development of reconfigurable systems.

2.1.1 MULTI-AGENT SYSTEM PLATFORMS

I. JADE

JADE (Java Agent Development Environment) is a FIPA compliant framework for the multi-agent software development. JADE is a high performance middleware for distributed agent system that is developed in the Java language. JADE is very efficient and flexible in terms of its communication architecture.

Following are the major features of JADE.

1. FIPA specifications compliant, includes the AMS (Agent Management System), the DF (Directory Facilitator), and the ACC (Agent Communication Channel).
2. Distributed platform that can be split on numerous hosts. Only one JVM is executed on each host. Agents are implemented as one thread and for actual and communications between agents, Java events are used. More efficient parallel tasks can be executed by JADE scheduler.
3. In order to implement multi-domain applications, multiple DFs (Directory Facilitator) can be started at run time.
4. FIPA97-compliant IIOP protocol to connect various unlike agent platforms.
5. In order to avoid marshaling and un-marshaling messages are transferred encoded as Java objects, rather than strings.
6. Ready to used library of FIPA interaction protocols is available.
7. Automatic registration of agents with the AMS.
8. Agents obtain their GUID (Globally Unique Identifier) from the platform at start-up using FIPA-compliant naming service.
9. GUI to manage various agents and agent platforms. The activity of each platform can be observed and recorded.

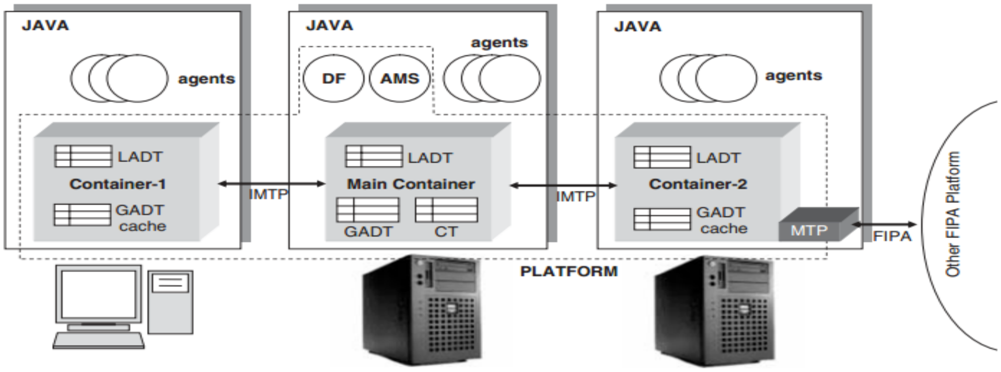


Figure 2.1: JADE Architecture (developing multi-agent systems with JADE, WILEY)

II. JIAC

JIAC is a multi-agent framework and runtime environment developed in JAVA, which incorporates the agent model with Service Oriented Architectures. JIAC's messaging infrastructure is based on ActiveMQ, facilitating transparent distribution of agents even beyond network boundaries. In JIAC new agents, services, as well as further agent nodes can be deployed at runtime. Agents can communicate with each other by means of service invocation, by sending messages to individual agents or multicast channels, and by complex interaction protocols.

Java Management Extension Standard (JMX) is used to remotely monitor and control the JIAC agents at runtime. Unlink JADE in JIAC the agents' behaviors and capabilities are implemented in so-called AgentBeans, controlled by the agent's life cycle. AgentBeans comply with the characteristics of software agents' because they are capable of reactive, proactive, iterative, or controlled behavior. Complete multi-agent system assembly is described in one or more Spring configuration files.

JIAC alows the likelihood of reusing applications and services, and notwithstanding adjusting them on runtime. The central purposes of JIAC are scalability, distribution, flexibility and autonomy.

The main features of JIACs framework are:

- Transparent Distribution

- SOA based interaction model
- Semantic service descriptions (ontologies based)
- Generic Security and Management
- Support for system reconfiguration in distributed environments

II. JADEX

Jadex is a Belief Desire Intention (BDI) reasoning engine that supports programming intelligent multi-agent software in Java. Jadex has very flexible reasoning engine and can be used with different middleware infrastructures such as JADE.

The main features of Jadex are:

- Component Programming Model
- Concurrency Model
- Multiple Interaction Styles
- Fast Prototyping
- Dynamic Reconfiguration
- Platform security model
- Technology Integration
- Simulation Support
- Overlay Network
- Tool Support

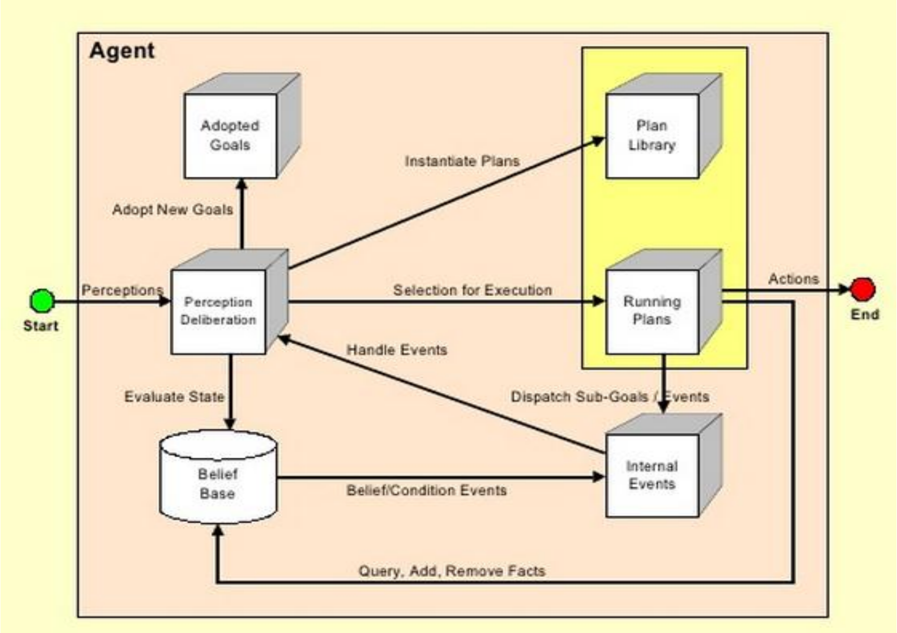


Fig 2.2: JADEX Architecture (Integrating multi-agent technology with cognitive modeling, by LeeRoy Bronner)

2.1.2 Using Design Patterns

Current state of the art of agent based system design and implementation focuses basically on message passing, exact specifications of ontology and interaction protocols identification. Such an approach is quite rigid and it is very difficult to design a system with re-configurable agent behavior. Using design patterns can be a way of system development with code reusability and encapsulation (REF1). To support use of design patterns in agent-based system development, it is emphasized (REF2) that efforts already made like Gaia (REF3), MaSe (REF4), and Tropos (REF5) focuses mostly on the designing of system goals, interaction, capabilities and so on, while the use of design patterns as recurring agent behaviors reduces code repetition. Despite this, design patterns likelihood to be reused with no evolution is low and their utilization in diverse context is not perfect. Additionally, the methodology of utilizing design patterns is not reconfigurable as indicated by changing system demands as they should be modified and agent code re-written.

2.1.3 USING STATE MACHINES AND AGENT SPECIFICATION LANGUAGE

To model the agent behaviors, some researchers have suggested the use of state machines (REF 6). Also to replace conventional programming language, Extensible Agent Behavior Specification Language (XABSL) has been devised (REF 7) that bolster the design of behavior module. XABSL specifications help create a transitional code and an agent device has been recommended that can run the created transitional code. Although XABSL can be used to model agent behavior, the generated code must be recompiled before the agent engine can execute it. Also the XABSL only models the behavior of a single agent and lacks the support of interaction among agents.

2.1.4 COSMOA

A multi-agent system COSMOA (REF8), that is designed in a way to aid the decision making process while carrying out a large scale medical response. The occurrences and the nature of the incidents are simulated and monitored by the system. On the basis of the information coming into the system, the number of casualties and injuries are predicted using heuristic approaches and alike methodologies. Thus the COSMOA makes the best use of ontology for the collection, integration and inference on heterogeneous data. So, making the ontologies the core of the multi-agent system, the systems are created in a way so as to be abstracted both for the behaviors and the internal characteristics. This approach defines agent behavior with respect to ontology layer. Agents develop their knowledge-base in order to define their behaviors by interacting with this layer of ontology. The ontology layer is variable in nature, depending on the problem domain. However, changing agent’s behavior in real-time is not possible in COSMOA.

Architecture of COSMOA is presented in figure 2.4.

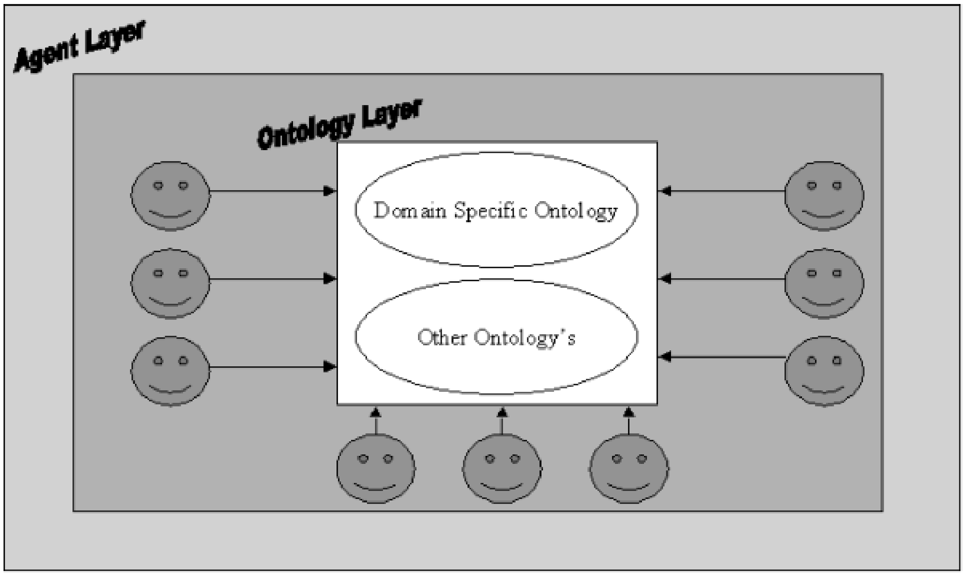


Fig- 2.4: Ontology-centric design (*European Journal on Artificial Intelligence - AI Communications*, Aug 2005)

2.2 RELATED WORK ANALYSIS

After analyzing current approaches, we found that the most of multi-agent development frameworks facilitate development of flexible intelligent multi-agent systems. Some of them are based on proprietary standards and specifically designed for specific environments so they are not easily implementable or interoperable with other systems. We also observed that some of them provide ease of development, and some facilitate the operation of large scale and distributed services. Some systems also allow the reusability of the applications & services and allow the modification during runtime.

After investigating current methodologies, we found that the majority of multi-agent development frameworks encourage development of adaptive smart multi-agent systems. Some of them are taking into account proprietary standards and particularly intended for particular situations so they are not effortlessly implementable or interoperable with different systems. We likewise watched that some of them give the simplicity of development, and some encourage the operation of substantial scale and distributed services. A few systems likewise permit the reusability of the applications & services and permit the modification at runtime.

However, no existing agent framework provides the luxury to re-configure a system in real-time, therefore it is needed to design an architecture that allows the system developers and domain experts to change the system in real-time without re-coding it. This will contribute to the development of systems that are reconfigurable at runtime and the already running system can be changed within no time

Chapter 3

Ontology centric Multi-agent system (OCMAS) Architecture

"I never did anything by accident, nor did any of my inventions come by accident; they came by work."
Thomas Edison

Multi-agent systems which have a large number of agents and frequently changing requirements at runtime have become a challenging problem for the developers to handle. Real-time reconfiguration of multi-agent systems has been the area not properly addressed over the time. Many agent system development frameworks have been around which ease the development process and support implementation/ deployment. These systems also allow for the possibility of reusing the system and modification at runtime, but none of these allow the real-time reconfiguration of the agent behaviors without recoding the application. When there is some change required in the behavior of the agent, the entire system is required to be brought down, re-coded, re-compiled, and redeployed affecting a number of agents running. A mechanism is required that can provide flexibility in the form of adaptability, modifiability and scalability for agent based system development through some development framework facilitating for real-time reconfigurable agent systems.

This work proposes an architecture, which will help minimize re-coding in case of changing requirements and will help avoid costly system redesigning with minimum performance compromises. Improvement in the information technology domain is making drastic changes by making devices more powerful in terms of computations, storage and memory capabilities so it has become easier for the organizations to adopt such architecture for agent system development and performance issue can be eliminated.

3.1 MULTI-AGENT SYSTEM MODELING:

Software developers have to frequently face the challenges of developing software whose requirements can change frequently in order to adapt to the required reconfigurations or other external factors. Different systems can have a number of ways to represent the same model, which introduces complications in the development and integration of the system. Any complete methodology for building multi-agent systems must address the interaction models used by the system and the internal behavior of individual modules in the system. Numerous methodologies have been proposed in the recent years, which concentrate on the development of powerful multi-agent systems by partitioning the construction of multi-agent systems into analysis, design, and implementation phases, however not very many of them really focus on designing the systems that are reconfigurable at real-time.

During the implementation, the models from the analysis and design phases of the system are used to transform the system into code. During the analysis phase in order to describe the system requirements, agents' roles are established and behaviors are attached to these roles. When system requirements are modeled in the analysis phases, then the designing phase of multi-agent system starts. During this phase the actual agent classes are created, and the described roles from the analysis phase are assigned to these agent classes.

Despite the work in the modeling of multi-agent systems, very few of them focused on designing some model to facilitate the ever changing requirements of the system. The system requirements can dramatically and frequently change in the life of the system. Many of these changes are of very small magnitude, but still require costly system, redesigning and re-coding. Therefore, some mechanism is required that can accommodate those changes without shutting down the system and coding the changes in the agent behaviors.

3.1.1 PROPOSED MODEL:

The reconfigurable agent framework architecture is divided into three major components. The first component consists of the agents that are interacting with each other by exchanging messages. The second component is the well-structured agent behavior ontology provides knowledge settings to the agents of the first component. Each behavioral setting is defined in the ontology. The third component consists of the well-structured and dynamic rules working as implementation units ready to be executed. This component facilitates the execution of the interaction component. Usually, only the knowledge in the second component is required to be updated with the changing requirements of the system at runtime. Table 3.1 explains the different components of the proposed model.

Interaction Model	A model that depicts the collaboration among multiple agents cooperating to accomplish a common goal.
Agent	Software unit that share common objective of the total system and has a responsibility to contribute to the realization of the system goal. The agent has a set of capabilities and the settings defined in the ontology which decides how an agent should act in a specific situation to achieve certain goals and what low-level computations it should invoke in the process.
Ontology Behavior Setting	A component that externalizes the agent knowledge. It is reconfigurable during runtime by an expert. Agents of the system use these settings to know and react to situation, decision making and interact with other agents. A collection of settings composes and define an agent's overall role. For any single agent, various settings can be defined in order to act differently in different situations.
Message	Information passed from one agent to another agent. Information in the form of string or any JAVA object encoded can be transmitted between sending and receiving agents. When a message is sent starting with one agent then onto the next, it depicts that the sending agent has performed its tasks and now the receiving agent has to perform its responsibilities, so that the overall system goal can be achieved.

Table 3.1

3.2 Architecture:

To support the proposed model, an architecture has been designed for real-time reconfigurable system and is shown in Figure 3.1. The architecture shows that there are two types of agents in the system, system agent, and domain agents. Apart from agents there are some other components as well, like the Rule Engine, Ontologies, and the Domain Expert. The domain agents are the agents responsible for executing the domain specific behaviors, such as for the prototype implemented in this research the auction is the domain and Buyer Agent, Auctioneer Agent, Shipment Agent etc. are domain agents. Each domain agent is created and managed by the System Agent. System Agent is also responsible to monitor the changes stored as settings in the ontology. Domain expert is the person who can make changes in the ontology in order to change agent behaviors. Rule Engine is responsible for executing the domain specific business rules. Rule engine supports rules, facts, conditions, and other related functions that eventually construct agent behaviors.

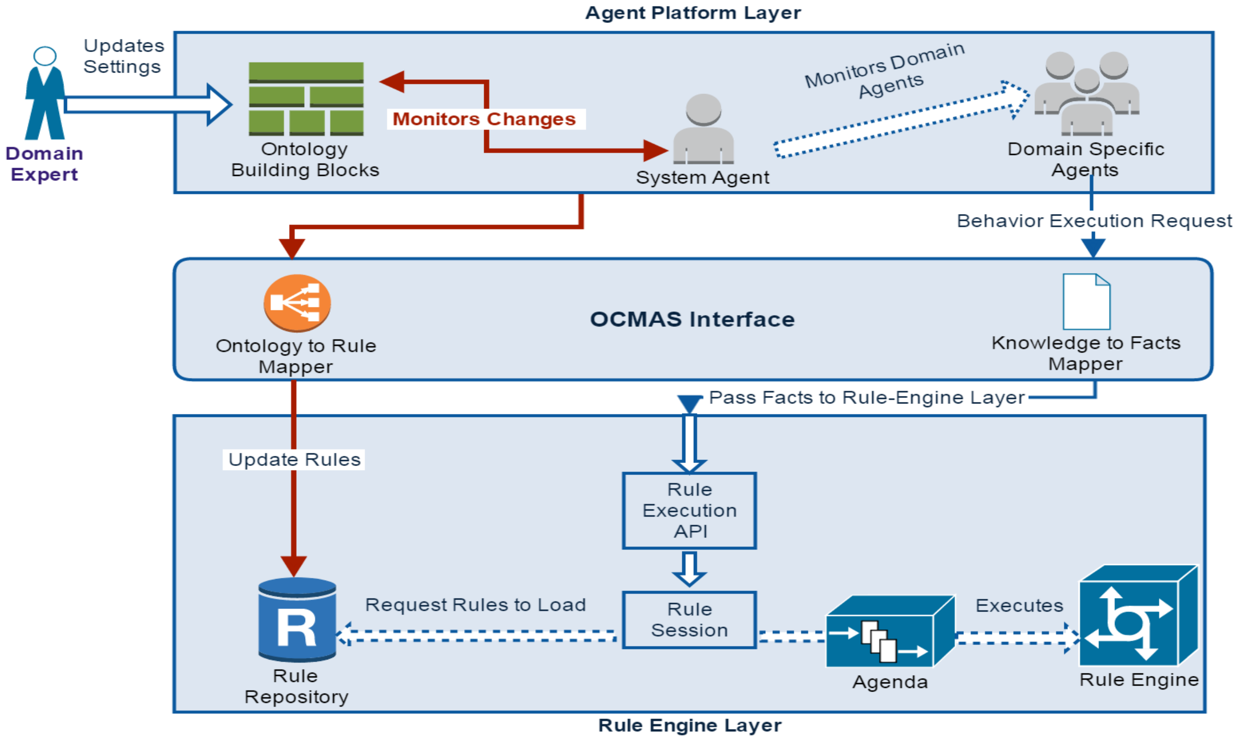


Figure 3.1: OCMAS Architecture

System agent registers and manages the domain agents. After registration, each Domain agent creates and invokes its session of rule engine by loading the set of required rules to work on. Each agent has a separate copy of rule-engine instance that it uses to execute behaviors by firing the applicable rules. Ontology Settings are the settings stored in the ontology as RDF triples in a well-defined structure that manipulates the rule agenda passed to the rule-engine as rule session. The rule engine then evaluates the rules in the light of rule agenda and facts passed to it by domain agents and execute the matching rules. Domain expert (DE) has complete knowledge about the implementation domain and understands the impact of the changes he makes into the ontology. DE can not only modify the settings to alter the existing behaviors of agents, but can also rearrange and add the settings to construct new behaviors from the existing behaviors.

3.2.1 MODELING ONTOLOGY-CENTRIC MULTI-AGENT SYSTEMS:

While defining the different components of the proposed architecture, now let's explain how possibly we can model the architecture. Here we divide the design of multi-agent systems guided by ontologies to control agent behaviors into two major models.

I. OPERATIONAL MODEL:

Operational models are designed to represent the agents, their association and domain logic in the form of settings stored in the ontology. Agents look into the ontology to read behavior settings and these settings guide the invocation of rules in the rule engine. This model then provides the foundation for the Agent Interaction Model. Each agent has a different set of capabilities within the business domain. Fig 3.2 depicts the agent system's operational model where each box represents an agent and inside each box the global variables and various settings like constraints, initializations, method invocations etc. are defined.

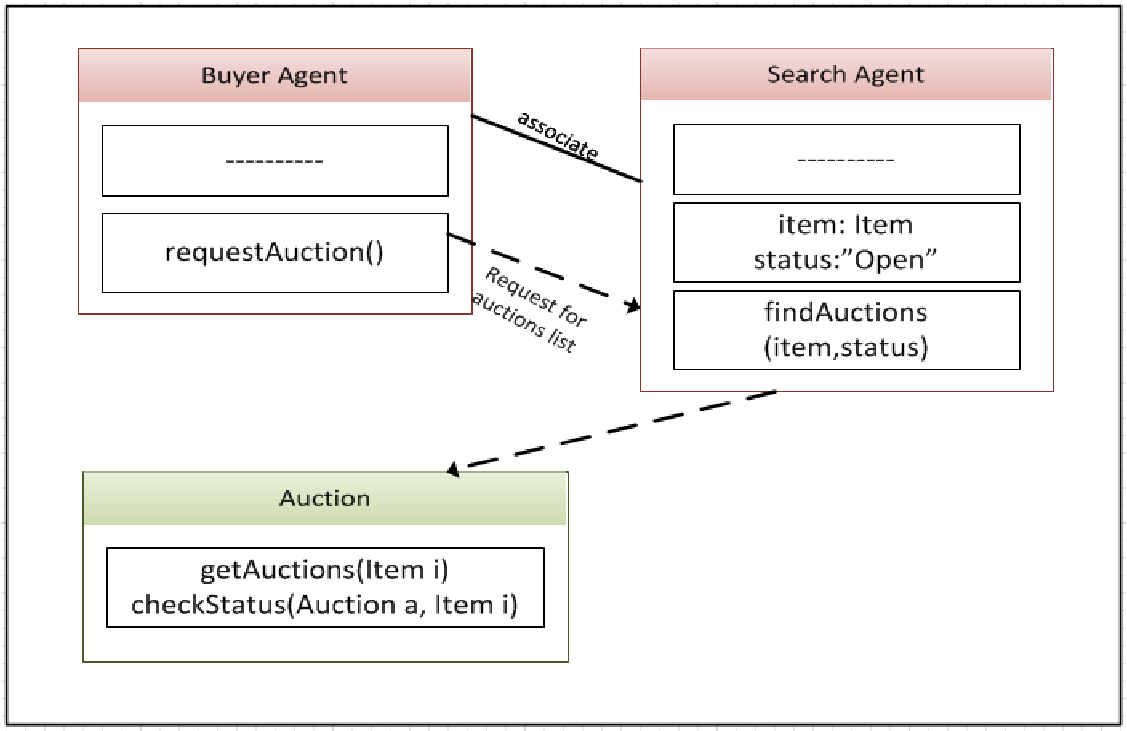


Fig 3.2 Showing agents operational model

For test case two agents "BuyerAgent" and "SearchAgent" are represented in Fig 3.2. "BuyerAgent" has a requirement to search for available auctions with a particular Item. For this the "BuyerAgent" has settings defined in the ontology as class instance "FindAuctions". Agent packages the item's information in a message and sends the message "msgSearchForAuctions" to the "SearchAgent". After receiving the message, the "SearchAgent" interacts with the auctioneers and return the information about the available auctions to the "BuyerAgent"

II. INTERACTION MODEL

When the operational model depicts the entire system on a whole and the association between various entities of the system, the agent interaction model is used to represent the collaboration among agents. This model is used to form agents, associated settings and messages around the required domain processes. Each agent is represented in this model along with the message it sends/receives to achieve specific business goal. To achieve real-time reconfiguration, the collaboration among the agents should be hidden at first because, unlike the conventional object oriented software models where each object knows in advance what other objects they will interact, in the proposed agent interaction model, the details of agent collaboration is kept hidden. Agents do not know in advance which other agents they will send messages to and receive messages from. The collaboration among agents is not hard coded within the system, so that agents can communicate with each other according to varying requirements.

3.2.2 EXECUTION APPROACH:

In object oriented system development, requirements are gathered and stored in documents. These requirements are then transformed into the system design and eventually machine executable code. Such a model is very important to successfully implement the required behavior. But these object oriented models become worthless with the lifetime of the running software because the system remains under continuous evolution and source code of the system is continuously updated. However the models initially developed are usually not redesigned. In OCMAS however, the system requirements are structured in the form of ontology settings and its realization is in the rule engine. These settings aren't only modelling the agent collaboration, but also the internal behavior of the agents. Since in OCMAS software model defined in the form of settings stored in ontology and this model is interpreted as agent behavior at runtime. The multi-agent system's implementation and design are closely integrated and whenever there is a change required in the system, it is made in the design model and system adopts changes seamlessly.

Agent's behavior, structured in the ontology can be formalized to achieve three major reconfiguration objectives. First is the reconfiguration of the internal behavior of every individual agent, secondly the interaction among agents, and thirdly the rules in the rule-engine that actually execute the agent behaviors. To explain the working of the proposed architecture, lets explain this with the help of the developed prototype of auction system. But before doing that's lets first explain why an auction system is a suitable option to select for such kind of work.

Of all the models analyzed, an auction model was found to be a standout amongst the best mechanisms for the implementation of the prototype. Among various auction protocols, the mostly used mechanisms are the English, Dutch, Vickery (Second price sealed offer auction), First-price sealed offer, and Double auction. Auction protocols can be named being either single-sided or two-sided. A single-sided auction is started by either the buyer or the seller, however a two sided auction is activated by both sellers and buyers, at whatever point in the midst of the executing period. The CDA is a case of a double-sided auction. The English auction, then again is a case of a single-sided auction. Table 3.2 and 3.3 show the differences between the auction protocols. The "Proportion of B-S" in table 3.2 signifies "Proportion of Buyers to Sellers" Table 3.1 demonstrates the diverse measurements of auction systems.

Auction Mode	One-sided(O): only bids or asks are permitted Two-sided (T): both bids and asks are permitted
Unit of Goods	One (O): only one good is auctioned during the auction Many (M): multiple goods are auctioned
Ratio of B-S	Many to one (MO): there are multiple buyers and only one seller One to many (OM): there is only one buyer and multiple sellers Many to Many (MM): there are multiple buyers and sellers
Information Revealed	Yes (Y): there is intermediate information revealed during the auction No (N): a bidder has no information about others.
Settlement Price	First price (F): highest price among all the bidders Second Price (S): second highest price among all the bidders.

Table 3.2

Table 3.3 demonstrates the differences between auction types in view of the measurements given in table 3.2.

Auction	Auction Mode		Unit of Goods		Ratio of B-S			Information Revealed		Settlement Price
	O	T	O	M	MO	OM	MM	Y	N	
English	✓		✓		✓	✓		✓		F
FPSB	✓		✓		✓	✓			✓	F
Vickery	✓		✓		✓	✓			✓	S
Dutch	✓		✓		✓	✓		✓		F
CDA		✓		✓			✓	✓		D

Table 3.3

In Dutch auctions, the supplier announces a high cost of an item, then the cost is persistently brought down until a buyer makes an offer. Klik-Klok [19] uses the Dutch auction protocol. In the sealed-bid auction, the offers are sealed i.e. the offers are not announced, i.e. not uncovered and members never know the affairs of their peers. The agents make the offer without having the information about other offers. In the First-price sealed offer auction the buyer with the highest offer is given the item though in the second-price sealed-bid auction (Vickery auction), the most noteworthy bidders pays the price equivalents to the value of the second most astounding bidder. “Timeshare Resale Internationals” [20] auctions of vacation are an illustration of the First-price sealed bid auction. “Antebellum Covers” [21] is an illustration of Second-Price sealed bid auction. Besides, the Dutch and FPSB auctions are deliberately identical. In both auction systems, the same bidding system is utilized by the buyers and bidders’ impetus is to offer lower than their valuations. In an English auction, an agent offers a bit higher than the present offer. This procedure proceeds until the customer's valuation has come or the bartering closures. In addition, the offers are uncovered to different bidders amid the process. Yahoo and eBay [22] auctions are cases of an English auction.

The auction protocols considered thus far have been single sided auctions. “Continuous Double Auctions (CDA)” is a two sided protocol. The buyers can submit offers and sellers can present the asking price to the seller whenever amid the executing period. The seller coordinates the offers and asks and afterward picks a price for transacting the item. Also, in CDA various merchandise are auctioned. This is unordinary however, as most auctions deal with a solitary good at once. It has likewise been seen from Table 3.2 that single sided auctions have both "MO" (multiple purchasers and one seller) and "OM" (one purchaser and various sellers) ration of B-S; this is because of the truth that these auctions can either be initiated by purchasers or by sellers.

3.2.3 ANALYSIS:

From the literature review, it has been found that the Dutch, the CDA, and the FPSB auctions support sellers more than buyers, though the Vickery auction may support buyers more than sellers (although not generally). In the event that the Dutch auction is considered for utilization in this research, then the client' may need to wait for the cost to fall until it achieves the buyer’s valuation. This may frequently wind up in a higher final value than can be defended on the grounds of supply and demand and may bring about a buyer to not purchase anything from the auction. The CDA favors sellers more than buyers in light of the fact that in the CDA clients are urged to offer higher because of prompt clearance of the market. In CDA, the time clearance is not known ahead of time which may bring about offering higher than the asset valuation. Subsequently, the CDA is not considered for application in this work.

The FPSB auction is additionally not by any stretch of the imagination suitable for utilization in this research. In an FPSB auction, a bidder can put one and only offer in the auction procedure which may bring about setting an offer much higher than the valuation of a thing. In the Vickery auction, the winner may put a high value bit if the second highest bidder puts an offer a great deal not exactly the most astounding bidder then advantage just goes to the customer. Who profits by a Vickery auction is subject to the second highest bidder. In the event that the second highest bidder offers low, then the advantage is just to the buyer. Accordingly the Vickery auction was discovered to be suitable. An English auction and Dutch auction are viewed as suitable for this work on the grounds that these two protocols are generally actualized and impressively diverse to one another. A system developed that can deal with such assorted qualities by changing starting with one auction mechanism, then onto the next will help us validate the research hypothesis.

3.2.4 WORKING

I. FORMALIZING AGENT BEHAVIORS:

Agent's behavior settings to perform tasks, evaluate conditions and response events are externalized as operational model in the ontology. This facilitates dynamic reconfiguration of agent's internal behavior. By defining changes in the <AgentFunctions> and <AgentFunctionsProcesses> classes and their instances of the ontology, the managed objects of the multi-agent system can be altered. In the example of the auction system, by making changes in the instances of <Agent Functions > and <AgentFunctionsProcesses> classes, new behavior can be constructed. Such that to transform system's behavior from the English auction to Dutch auction, settings in the ontology can be arranged in a particular combination so that the new behavior is constructed i.e. Dutch in this case. These changes will take effect in real-time as soon as they are made.

A single function from the buyer's ontology represented here in the form of options states that can be toggled to instruct the rule engine how to execute rules and define behaviors at runtime. The differences in settings for English and Dutch auctions are highlighted.

<functions>	<functions>
<buyer>	<buyer>
<function name="Bid To Auction">	<function name="Bid To Auction">
<state value="ON" />	<state value="OFF" />
<processes>	<processes>
<process name="Log" state="ON" />	<process name="Log" state="ON" />
</processes>	</processes>

</function>	</function>
<function name="Make Payment">	<function name="Make Payment">
<state value="ON" />	<state value="ON" />
<processes>	<processes>
<process name="Log" state="ON" />	<process name="Log" state="ON" />
</processes>	</processes>
</function>	</function>
</buyer>	</buyer>
</functions>	</functions>

For a complete behavior change between English and Dutch auctions, detailed settings are represented in **Appendix A**,

II. ADAPTING TO CHANGE

Domain concepts are defined in the ontology and the structure of the ontology is fixed. These concepts must conform to the structure of the ontology. Whenever there is a need of some new domain concepts which currently do not exist, it can be defined as a new instance and associated properties. These new concepts will become available to the rule-engine for processing and invocation by the running multi-agent system. A change monitor implemented as a thread is responsible to captures changes in the ontology and then, it applies those changes in the rule engine's knowledge-base. Agents interact with peer agents by sending and receiving messages. When the sender agent sends a message to the receiver agent, the receiving agent processes the message by using its internal objects and performs the required computations. During processing different circumstances may arise that agents need to respond by looking into the changed settings in the ontology. The settings defined in the ontology enable the agent to react in a particular situation. Multiple settings can be defined in ontology for a particular behavior to achieve various goals. By defining model for agent interaction and defining settings for collaboration helps to achieve re-configuration. Agent's interfaces are structured in the ontology and also the functionality that it provides. The structured interface ensures re-configuration by introducing agreement between the interacting agents. By using such a model, the re-configurable agent model not only provides adaptation externally by using structured interfaces, but also the internal behaviors realized in the rule engine. In this way the agent system is reconfigurable both internally and externally.

3.3.1 PROTOTYPE TECHNOLOGIES SELECTION:

The component arrangement, using the suggested approach is depicted in Fig 3.3. At the core of the architecture is the multi-agent auction system developed using the JADE platform. Then a central ontology to structure agent’s behavioral settings is deployed. This ontology contains the definitions of each and every behavior setting of agents. The ontology parsing module is implemented as a mapping component that is responsible for fetching the settings stored in the ontology and providing these settings as knowledge input to the rule engine. This component uses the SPARQL query language to fetch settings from the ontology. The ontology settings are continuously updated by the domain experts to adjust agents’ behavior at runtime. A system agent responsible for monitoring changes in the ontology and controlling and creating the domain agents looks into the ontology to get the most recent settings. When there some change in the settings, the system agent updates the rules in the rules repository. These rules are then made available to the agents for behavior invocations.

For the purpose of prototype, DROOLS rule-engine has been used. “Drools is a business rule management system (BRMS) with a forward and backward chaining inference based rules engine, more correctly known as a production rule system, using an enhanced implementation of the Rete algorithm” [23]. Initially, each agent has basic knowledge required to work in an environment. During runtime of system, the requirements can always be changed and each and every agent in the system adapts to the changed settings in knowledge. As a result every agent has the required behaviors as soon as they are specified. The reason behind the selection of JADE as the multi-agent framework and Drools as the rule-engine is that these two are the current state of the art and also the industry standard open source products. However the proposed ontology-centric model is not specific for these technologies. Any multi-agent framework with BDI model can be used for implementation. Also the choice of selecting a rule-engine for implementation is open.

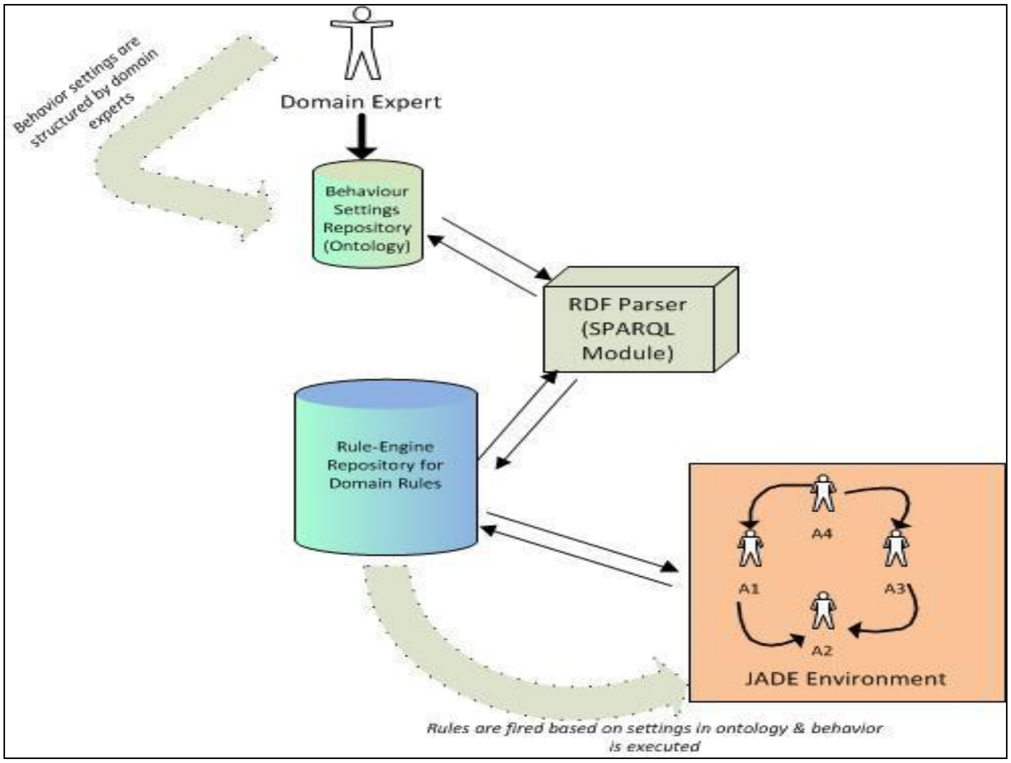

Behavior settings are structured by domain experts
Domain Expert
Behaviour Settings Repository (Ontology)
RDF Parser (SPARQL Module)
Rule-Engine Repository for Domain Rules
A4
A1
A3
A2
JADE Environment
Rules are fired based on settings in ontology & behavior is executed

Fig 3.3 Prototype Technologies

In this implementation, whenever the change is made in the ontology, the system agent updates the rules in the rule-engine. A mechanism of "CONTROL & PROCESS" is used here to control and update the rules. In the light of ontology the CONTROL part of rule decides whether a particular rule is applicable in a particular situation or not. And the PROCESS part decides what processes are applicable in a rule to execute a behavior. An example of a rule from the Auctioneer Agent's repository is shown here in Figure 3.4.

```
rule "Offer Accepted"
when

$a:DroolsAuction(leadingBuyer!=null,accepted==true,proposing==false,won==false)
    /*control*/eval(true)
then
    /*process*/auctioneer.logRule("Offer Accepted")
    $a.setProposing(true)
    /*process*/auctioneer.requestEmail($a.getLeadingBuyer(),"You have won the 
auction "+$a.getId(), $a.getWinningMailable())
    auctioneer.won($a)
end
```

Fig 3.4 DROOLS Rule

This approach makes the model flexible and adaptable to changes without having to make changes in the application code.

3.4 OCMAS ARCHITECTURE FEATURES:

The OCMAS Architecture Provides more control over the system to Administrators and domain experts in order to reconfigure the agents behaviors It minimizes management complexities for the programmers and system analysts to recode the system when a change is required. Here agents' behaviors are determined in the ontology and translated by agents as behavioral rules during the system execution. Existing behaviors of agents can not only be changed, but new behaviors can also be constructed from the existing behaviors. OCMAS provides an alternative against conventional multi-agent systems development where the number of agents is too large by cutting costs of re-coding, maintenance and re-deployments. Also settings arrangement to construct behaviors is easy to read and understand from the domain experts, and changes can be easily be made with the continuously changing requirements. This model makes the utilization of existing infrastructure and technologies. Along with other features mentioned above, the proposed model is also scalable and adaptable.

Chapter 4

Evaluation & Results

"Mathematics is the science which draws necessary conclusion"
Benjamin Peirce

The earlier chapters explained the methodology used to address the identified research problem. The literature review has been explained in Chapter 2 which helped to answer the question of "Why using an ontology centric approach is required to define the behavior of an agent?" After the literature review, the ontology-centric approach to structure agent behavior was proposed and explained in Chapter 3. In order to evaluate the proposed model and to provide proof of concept, a prototype system was developed. The implementation detail of the system was also explained in Chapter 3 which helped to answer our second research question that "How an ontology centric approach help define the behavior of an agent at real-time?" To answer our third important question of research, it is essential to analyze that "How successfully the developed system can be reconfigured at real-time". As evaluation plays a significant role in the assessment of an idea, thus to evaluate the effectiveness of the proposed model, a mechanism is required. Furthermore, this will also help us to evaluate the research hypothesis mentioned in Chapter-1.

This chapter explains how the complete evaluation process has been carried out in a real-world environment. In order to find out how effectively the designed system adopts changes at real-time, it is essential to look into the system's ability to reconfigure its behavior at real-time whilst at the same time providing the desired behavior. For this purpose, system has been assessed with different number of agents and distinct combination of settings. It is also imperative to understand how scalable the system is under different scenarios and what is the level of stability. To achieve this some comprehensive testing mechanism is required to evaluate the system with such magnitude and very little previous work. For this purpose a system is required to be implemented as a case study. This case study will help us define the behavioral and syntactical boundaries for the multi-agent system developed using the proposed approach. The main characteristics of multi-agent system that should be considered for testing the proposed model should be that: (i) each agent in the system has a range of behaviors and these behaviors can be either distinct or similar in concept. (ii) These agents should be able to communicate and respond to each other. (iii) Agents should have vital elements, like initial belief, goals, triggering events, contexts, etc.

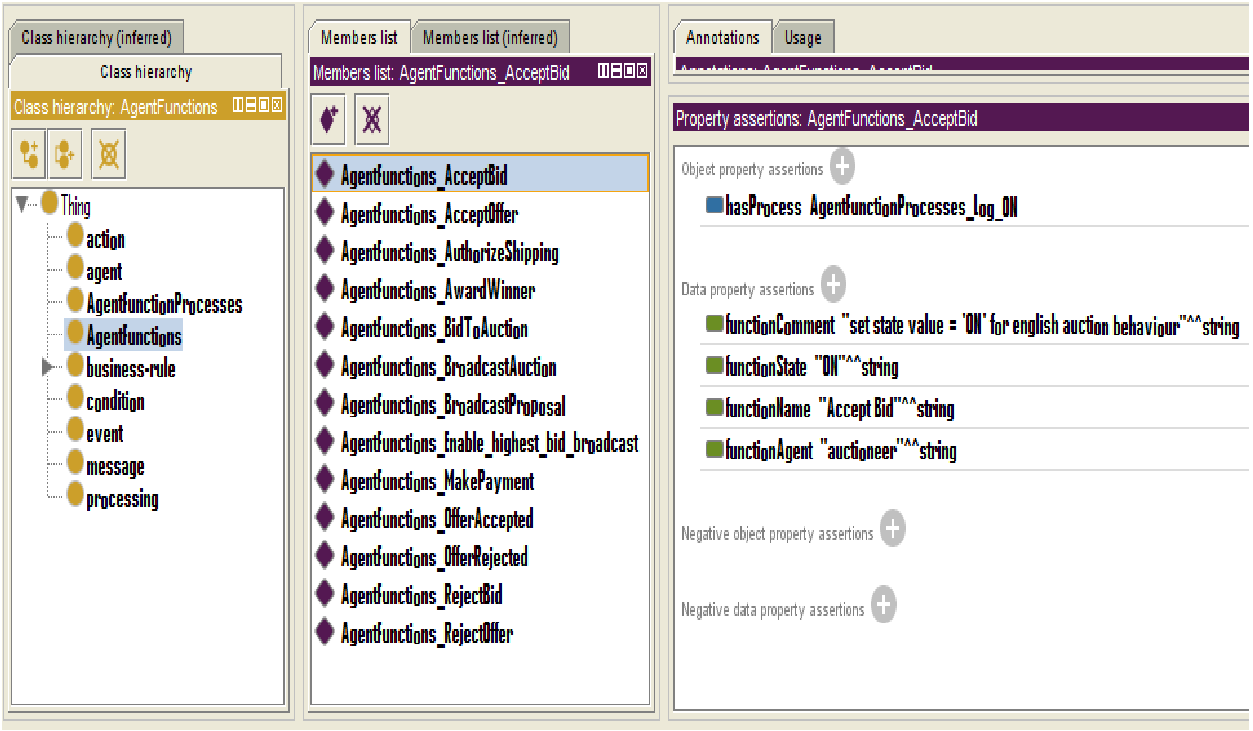
An auction system has been selected to evaluate the proposed idea. As auction is a process of buying and selling goods or services by offering them up for bid, taking bids, and then selling the items to the highest bidders. An agent based online auction system is a multi-agent system that comprises software agents to handle tedious tasks on behalf of users with minimum response time. Therefore the reason behind selection of such system to be developed as a multi-agent system is that, in auction systems when the number of users and products increase, more time is required for a user to search and bid for an auctioned item. Another key reason for choosing an auction system is the auction protocols. An auction can be distinguished by a number of auction protocols. For example English auction, Dutch auction, Japanese auction, etc. An English auction is the open-bid ascending price auction in which the price of the auctioned item is gradually raised and then after specified time, the highest bidder wins the auction. Whereas in a Dutch auction, the protocol is different in which the auctioned items start at a high price which is gradually decreased. When the price reaches a convenient threshold for one of the participants in the auction, it will call out and receives the goods in exchange for the current price. So when kept the proposed model with the auction systems, it seemed a good idea to evaluate the system against this auction protocol variation where designed system can transform from one protocol implemented as agent behaviors to another protocol without re-coding the system. This chapter helped to evaluate that how successfully an auction system with agent behavior set to English auction can be reconfigured to Dutch auction or any other auction protocol.

4.1 QUALITATIVE EVALUATION:

To test the ability of the developed prototype to reconfigure the system and ontologies in real-time and to evaluate the benefits of using such approach in the development of multi-agent systems, we implemented an auction system using the OCMAS model. The rest of this section discusses the experiences of using OCMAS based on the design of the sample system. The creation of the system is based on the structured ontology settings defined before the start of the development process. The steps in the development of the auction system start with the construction of the ontology. By handling ontology creation right before the actual implementation, the methodology allows to analyze the requirements thoroughly before creating the implementation model. Designers can determine exactly how to arrange different components of the system which are necessary for the system development.

After some deliberation, it was thought that with the integration of settings stored in the form of ontologies and rule engine based facts, the developed system will be more stable and flexible while agent behaviors are mutable. On alterations of the settings modeled in ontology, an agent with its abilities and behavior execution requirements can be transformed into an agent as an autonomous software unit providing services in a totally different and adaptable manner. A relationship between agent activities to fulfill a certain requirement is represented as a set of option arrangement through which these software units cooperate and act to achieve a certain goal. An internal relationship inside an agent in behavior requirements denotes that there is an internal invocation in such a software unit represented in the ontology. Since each agent in the prototype has corresponding settings defined in the ontology, its behaviors and its runtime mappings are modeled together closely. Such an approach not only costs much less effort when we develop them, less risky when we need changes in the behavior and more easy to maintain consistency between both states. In other words, agents' behaviors and their realization are integrated seamlessly during the development and their architecture fits each other. Ontologies can help model agent behavior substantially, especially when the designed system is complicated and distributed. Domain knowledge can be reflected and fit well in the structured ontology. In the prototype system, each and every functionality of the agent has matched perfectly well with the setting elements described in the ontology which is correlated with other functional elements in the ontology, reflecting its function definition semantics. Such behavioral mapping brought us benefits in several aspects.

Agent activities relationship for certain requirements represented in figure 4.1 as a set of function options with ON/OFF state through which the system decides how to execute behaviors at runtime.


Class hierarchy (inferred)
Class hierarchy
Class hierarchy: AgentFunctions
Thing
action
agent
AgentFunctionProcesses
AgentFunctions
business-rule
condition
event
message
processing
Members list
Members list (inferred)
Members list: AgentFunctions_AcceptBid
AgentFunctions_AcceptBid
AgentFunctions_AcceptOffer
AgentFunctions_AuthorizeShipping
AgentFunctions_AwardWinner
AgentFunctions_BidToAuction
AgentFunctions_BroadcastAuction
AgentFunctions_BroadcastProposal
AgentFunctions_Enable_highest_bid_broadcast
AgentFunctions_MakePayment
AgentFunctions_OfferAccepted
AgentFunctions_OfferRejected
AgentFunctions_RejectBid
AgentFunctions_RejectOffer
Annotations
Usage
Annotations: AgentFunctions_AcceptBid
Property assertions: AgentFunctions_AcceptBid
Object property assertions
Data property assertions
Negative object property assertions
Negative data property assertions

hasProcess AgentFunctionProcesses_Log_ON

functionComment "set state value = 'ON' for english auction behaviour"^^string

functionState "ON"^^string

functionName "Accept Bid"^^string

functionAgent "auctioneer"^^string

Figure 4.1: An ontology representing settings along with their states for the English Auction

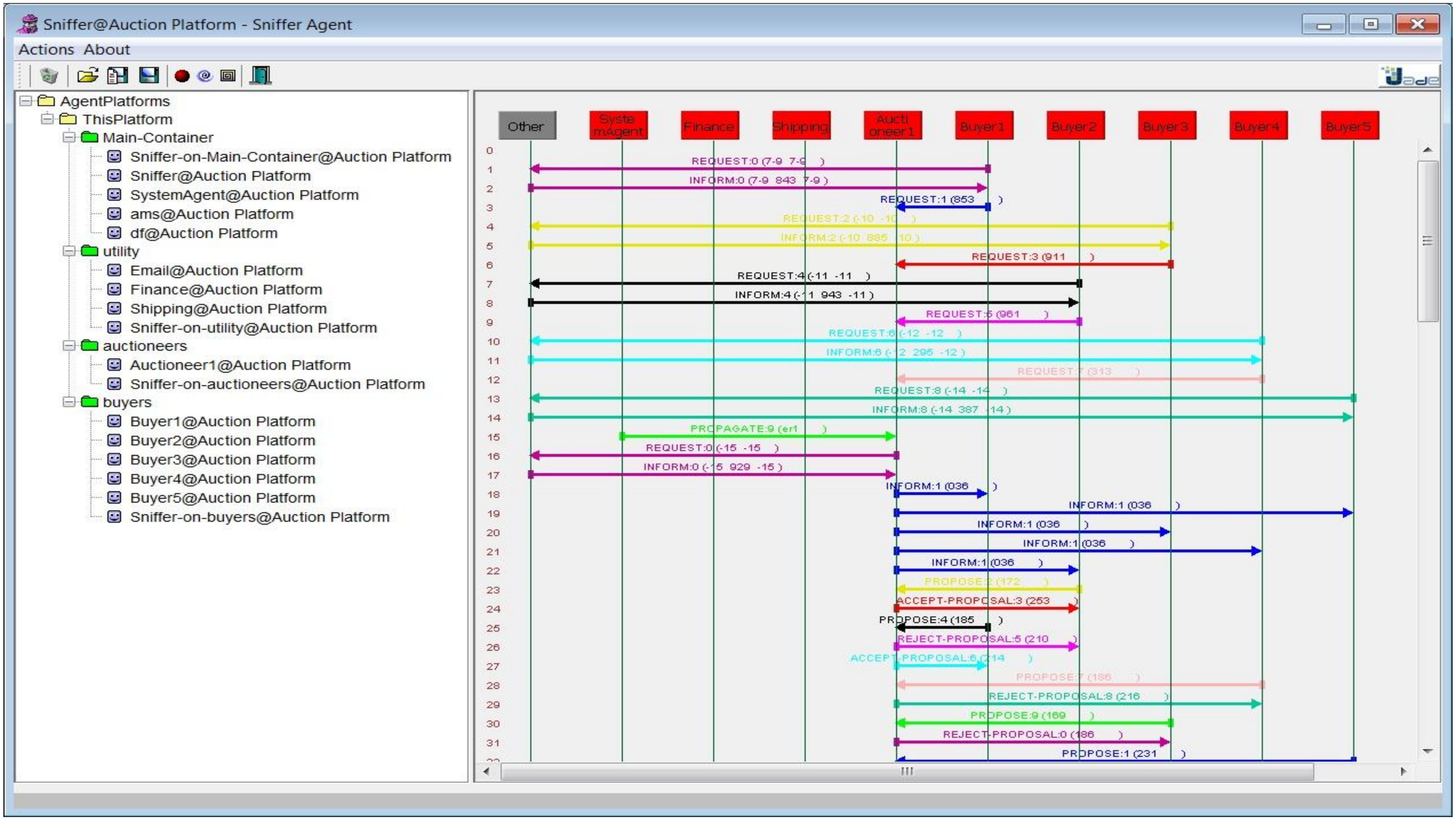
4.1.1 ACCURACY OF THE PROTOTYPE:

Accuracy is the first parameter which is to be considered during the evaluation of the system because the developed system might be verified as entirely correct but the results can be extremely inaccurate if the designed model is inadequately applied. The complete model needs to be evaluated for correctness to ensure that it is appropriate and well coded. Properly designed model along with the added dimension that the developed system should adapt to changing behaviors, is an important metric to evaluate for quality results. In the proposed model it is estimated in two possible ways. Firstly, it is interpreted manually through the possible transactions that should carry out during the auction process and their results. Secondly, it is analyzed using the prototype to determine the system's performance and outcome. At the end a comparison is made between the two outputs. The manual assessment of the process shows that how correctly, we can map the process with the automated system when given the large number of options and when the process is reconfigured. The manual assessment of auction behavior can be done in a number of ways because many factors affect this behavior like the auction duration, buying preferences or methods, etc. In order to explore that, the assessment is made with the standard documented process and results are evaluated by the automated system to ensure the correctness of the system, assuming that there is no 'Sniping' and 'Shilling' in bidding behaviors.

Following steps are involved in the implementation of English auction.

i. Creation of Auction
ii. Setting up reserve price, start time, end time, quantity and other relevant parameters.
iii. Announcement of auction for the buyers.
iv. Buyers bid for an auctioned item of interest.
v. Bid is evaluated against some rules and if the bid is a valid offer, it is accepted, else rejected.
vi. After some certain amount of inactivity from bidders or when the time of the auction is expired, the item is sold to the highest bidder.
vii. Payment is received from the winning buyer
viii. Shipment is made.

So when tested the developed system with the above mentioned English auction procedures, the results were according to the English configuration settings and expected output. The system output and the messaging between the agents are represented in the *Figure 4.2* given below.



- AgentPlatforms
  - ThisPlatform
    - Main-Container
      - Sniffer-on-Main-Container@Auction Platform
      - Sniffer@Auction Platform
      - SystemAgent@Auction Platform
      - ams@Auction Platform
      - df@Auction Platform
    - utility
      - Email@Auction Platform
      - Finance@Auction Platform
      - Shipping@Auction Platform
      - Sniffer-on-utility@Auction Platform
    - auctioneers
      - Auctioneer1@Auction Platform
      - Sniffer-on-auctioneers@Auction Platform
    - buyers
      - Buyer1@Auction Platform
      - Buyer2@Auction Platform
      - Buyer3@Auction Platform
      - Buyer4@Auction Platform
      - Buyer5@Auction Platform
      - Sniffer-on-buyers@Auction Platform

```
0
1  REQUEST:0 (7-9 7-9 )
2  INFORM:0 (7-9 843 7-9 )
3  REQUEST:1 (853 )
4  REQUEST:2 (-10 -10 )
5  INFORM:2 (-10 886 -10 )
6  REQUEST:3 (911 )
7  REQUEST:4 (-11 -11 )
8  INFORM:4 (-11 943 -11 )
9  REQUEST:5 (961 )
10 REQUEST:6 (-12 -12 )
11 INFORM:6 (-12 295 -12 )
12 REQUEST:7 (313 )
13 REQUEST:8 (-14 -14 )
14 INFORM:8 (-14 387 -14 )
15 PROPAGATE:9 (er1 )
16 REQUEST:0 (-15 -15 )
17 INFORM:0 (-15 929 -15 )
18 INFORM:1 (036 )
19 INFORM:1 (036 )
20 INFORM:1 (036 )
21 INFORM:1 (036 )
22 INFORM:1 (036 )
23 PROPOSE:2 (172 )
24 ACCEPT-PROPOSAL:3 (253 )
25 PROPOSE:4 (185 )
26 REJECT-PROPOSAL:5 (210 )
27 ACCEPT-PROPOSAL:6 (214 )
28 PROPOSE:7 (186 )
29 REJECT-PROPOSAL:8 (216 )
30 PROPOSE:9 (169 )
31 REJECT-PROPOSAL:0 (186 )
   PROPOSE:1 (231 )
```

Fig 4.2: Sniffer results for messages exchanged between agents in English Auction

The systems results extracted from messages exchanged between agents are represented here in Table 4.1

Auctioned Item	Mobile-A600	Starting Price	Rs 9500/-
Quantity	**1**	**Duration**	**5 minutes**

Auctioneer Agent	BuyerAgent1	BuyerAgent2	BuyerAgent3	BuyerAgent4	BuyerAgent5
Proposed starting price 9500	*Received Proposal-Rs.9500*	*Received Proposal-Rs.9500*	*Received Proposal-Rs.9500*	*Received Proposal-Rs.9500*	*Received Proposal-Rs.9500*
ReceivedBid-Rs.9924				*BidSent-Rs.9924*	
ProposedOffe-Rs.9924	*Received Proposal-Rs.9924*	*Received Proposal-9924*	*Received Proposal-Rs.9924*	*Received Proposal-Rs.9924*	*Received Proposal-Rs.9924*
ReceivedBid-Rs.9954			*BidSent-Rs.9954*		
ProposedOffer-Rs.9954	*Received Proposal-Rs.9954*	*Received Proposal-Rs.9954*	*Received Proposal-Rs.9954*	*Received Proposal-Rs.9954*	*Received Proposal-Rs.9954*
ReceivedBid-Rs.10358	*BidSent-Rs.10358*				
ProposedOffer-Rs.10358	*Received Proposal-Rs.10358*	*Received Proposal-Rs.10358*	*Received Proposal-Rs.10358*	*Received Proposal-Rs.10358*	*Received Proposal-Rs.10358*
ReceivedBid-Rs.11047		*BidSent-Rs.11047*			
ProposedOffer-Rs.11047	*Received Proposal-Rs.11047*	*Received Proposal-Rs.11047*	*Received Proposal-Rs.11047*	*Received Proposal-Rs.11047*	*Received Proposal-Rs.11047*

Auctioneer Agent	**BuyerAgent1**	**BuyerAgent2**	**BuyerAgent3**	**BuyerAgent4**	**BuyerAgent5**
ReceivedBid-Rs.12371					*BidSent-Rs.12371*
ProposedOffer-Rs.12371	*Received Proposal-Rs.12371*	*Received Proposal-Rs.12371*	*Received Proposal-Rs.12371*	*Received Proposal-Rs.12371*	*Received Proposal-Rs.12371*
ReceivedBid-Rs.12662	*BidSent-Rs.12662*				
ProposedOffer-Rs.12662	*Received Proposal-Rs.12662*	*Received Proposal-Rs.12662*	*Received Proposal-Rs.12662*	*Received Proposal-Rs.12662*	*Received Proposal-Rs.12662*
ReceivedBid-Rs.13657			*BidSent-Rs.13657*		
ProposedOffer-Rs.13657	*Received Proposal-Rs.13657*	*Proposal-Rs.13657*	*Proposal-Rs.13657*	*Proposal-Rs.13657*	*Proposal-Rs.13657*
ReceivedBid-Rs.14032				*BidSent-Rs.14032*	
ProposedOffer-Rs.14032	*Received Proposal-Rs.14032*	*Received Proposal-Rs.14032*	*Received Proposal-Rs.14032*	*Received Proposal-Rs.14032*	*Received Proposal-Rs.14032*

Auction Won by BuyerAgent-4 ***Sending Acknowledgement***				***Message:”Won Auction.*** ***Price: Rs.14032 item:MobileA600”***	

Table 4.1: Output from English Auction Implementation

Steps involved in a typical Dutch auction are:

i. Creation of Auctions.

ii. Setting up of very high starting price for the item, start time, end time, quantity and other relevant parameters.

iii. Announcement of auction for the buyers.

iv. Auctioneer proposes the price to the participating buyers.

v. After a certain amount of time, if none of the buyers are interested in the offer, the value is decreased to some value and offer is made again to the buyers and this process continues until some buyer is interested in the offer.

vi. If some buyer is interested in the offer, it accepts the bid.

vii. Payment is received from the winning buyer.

viii. Shipment is made.

Here after making the changes in the ontology at runtime according to the Dutch auction process and tested the developed system, the results were exactly according to the Dutch configuration settings. The system output and the messaging between the agents are represented in *Figure 4.3*.

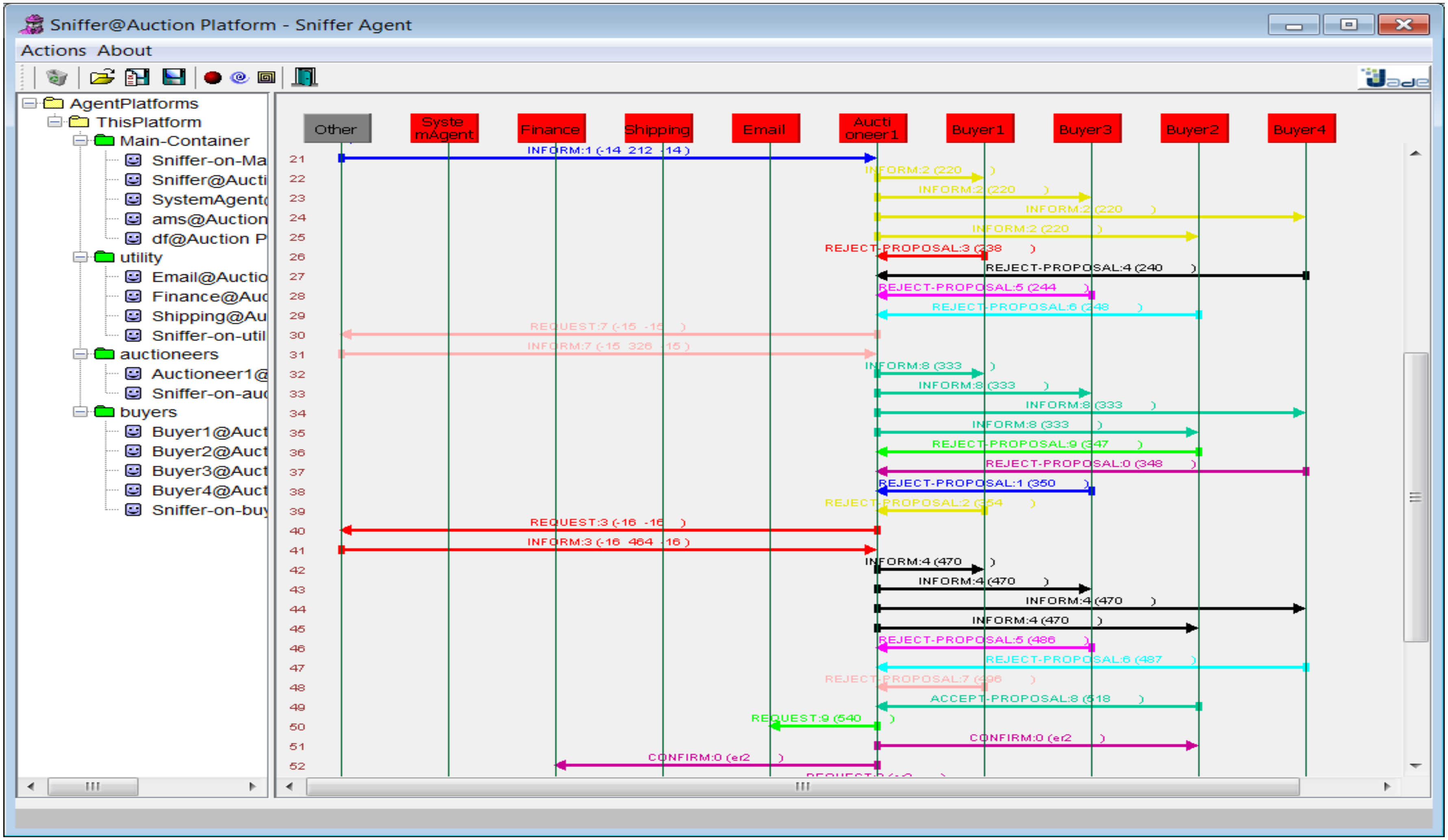


Fig 4.3: Sniffer results for messages exchanged between agents in Dutch Auction

The systems results extracted from messages exchanged between agents are represented here in Table 4.2

Auctioned Item	**Mobile-A600**	**Starting Price**	**Rs 22,500/-**
Quantity	**1**	**Duration**	**5 minutes**

Auctioneer Agent	**BuyerAgent1**	**BuyerAgent2**	**BuyerAgent3**	**BuyerAgent4**	**BuyerAgent5**
Proposed 22500 to all buyer agents.	*Not-Interested*	*Not-Interested*	*Not-Interested*	*Not-Interested*	*Not-Interested*
OfferRevised to 21900					
Proposed 21900 to all buyer agents.	*Not-Interested*	*Not-Interested*	*Not-Interested*	*Not-Interested*	*Not-Interested*
OfferRevised to 20540					
Proposed 20540 to all buyer agents.	*Not-Interested*	*Not-Interested*	*Not-Interested*	*Not-Interested*	*Not-Interested*
OfferRevised to 19876					
Proposed 19876 to all buyer agents.	*Not-Interested*	*Not-Interested*	*Not-Interested*	*Not-Interested*	*Not-Interested*
OfferRevised to 18284					
Proposed 18284 to all buyer agents.	*Not-Interested*	***I-am-Interested***	*Not-Interested*	*Not-Interested*	*Not-Interested*

Auctioneer **Agent**	**BuyerAgent1**	**BuyerAgent2**	**BuyerAgent3**	**BuyerAgent4**	**BuyerAgent5**

Auction Won by BuyerAgent-2 ***Sending Acknowledgement***		***Message:”Won Auction.*** ***Price:18284 item:MobileA600”***			

Table 4.2: Output from Dutch auction Implementation

4.2 QUANTITATIVE EVALUATION:

For the evaluation of the proposed model, a test scenario is created in which an auction system developed using the proposed model is compared with an identical auction system developed without using the OCMAS model. Both the systems were tested under different workload which includes multiple numbers of auctions and different number of agents.

The workload datasets were categorized into five major categories which are smaller, small, medium, large and larger. Each smaller dataset contained tests from a maximum of 100 agents and from 1-5 parallel auctions. Similarly for small, medium, large and larger datasets the number of agents was 250, 500, 1000 and 2000 respectively. A minimum of 100 and a maximum of 2000 agents were considered to keep the evaluation manageable. These different datasets facilitate to determine how the efficiency and productivity of the system varies under different tasks at hand. Also the change in scenario reveals the possible variations that can be given to the system in real environment. For example, it may be possible that the buyer budget is not enough to get an item at a higher price so buyers increase bids in a controlled manner and auction process may take longer. The complete detail of experiments and associated results are covered in the next sections.

EXPERIMENTAL DESIGN

Two prototypes of an auction system were developed. In order to evaluate the performance and scalability of the proposed approach, an equally identical auction system was developed using the JADE framework. Both prototypes were evaluated using same result metrics. These results were analyzed by gradually increasing the number of agents in each system. The most significant measures for quantitative evaluation are the CPU and memory utilizations by the software. The CPU and memory usage patterns are used to compare the performance and scalability features of both prototypes.

I. Memory Utilization

The memory utilization is what the system believes is memory currently being actively used by the application. This is an estimate calculated by a form of statistical sampling and this statistical sampling will most definitely come in handy when doing capacity and scalability planning. Memory utilization is in our opinion the memory that should be used to analyze trends, monitor capacity, etc. Evidently almost every capacity planning or monitoring tool out there uses this metric for evaluation. When benchmarking the memory utilization, it is observed that the proposed approach consumed more memory than the system developed using conventional JADE approach. The major difference in the memory utilization occurred at the time of agents' creation when each agent looked into the ontology for the settings and at the time of initialization when agents invoked an instance of the rule engine knowledge-base. This knowledge-base invocation helps the agents to build knowledge sessions during execution. Since DROOLS rule engine is used in the implementation of this prototype and it is really a heavy component, therefore the difference in the memory utilization is somewhat obvious. Secondly, there is also difference in memory consumption during the execution of auction process but that is not much considerable. Since in the proposed model, the agent behavior implementation is controlled using the rules and rules execution avail memory. Again the additional memory use is by the DROOLS rule engine. The use of memory depends on the rules. A large number of objects can be reasonably processed if given enough memory because it has to load the RETE network in memory, so memory usage is a multiple of number of objects – i.e. space for objects + space for network structure, indexes etc.

However, such an approach can be replaced in future by using the DROOLS server, which is a pretty new module within the Drools, which can be adopted to execute knowledge-base remotely using HTTP through REST and SOAP interfaces. An added advantage of using Drools Server would be that, it overcomes the platform dependency and can be invoked by non-Java applications as well.

Here is the *Figure 4.4* representing the memory consumption of the prototype developed using the proposed model in comparison to the one developed without it.

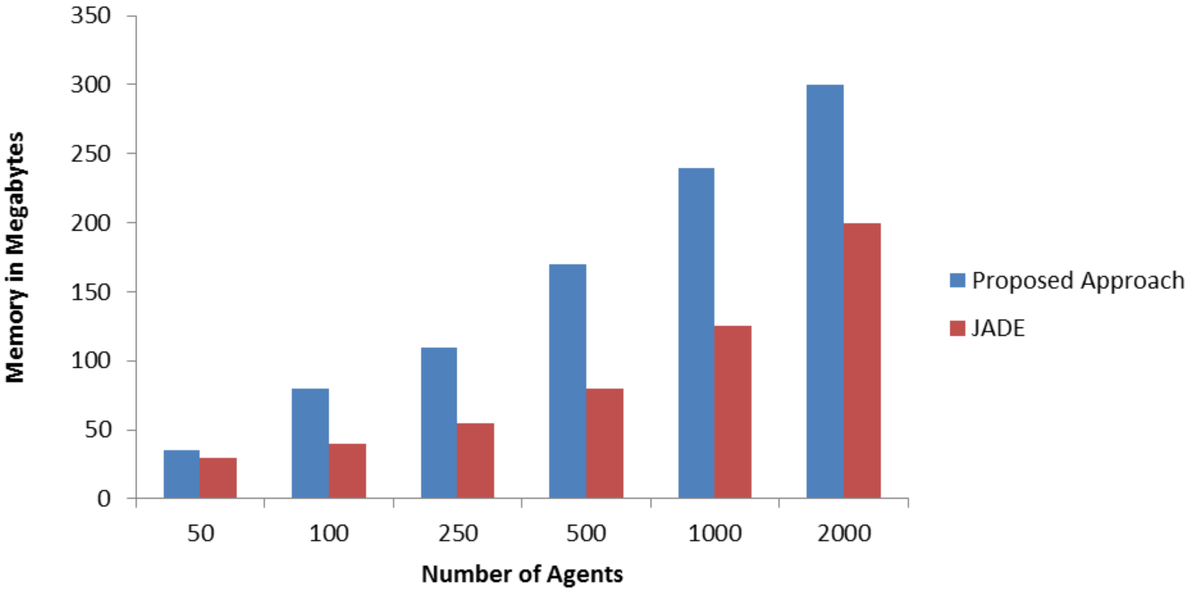


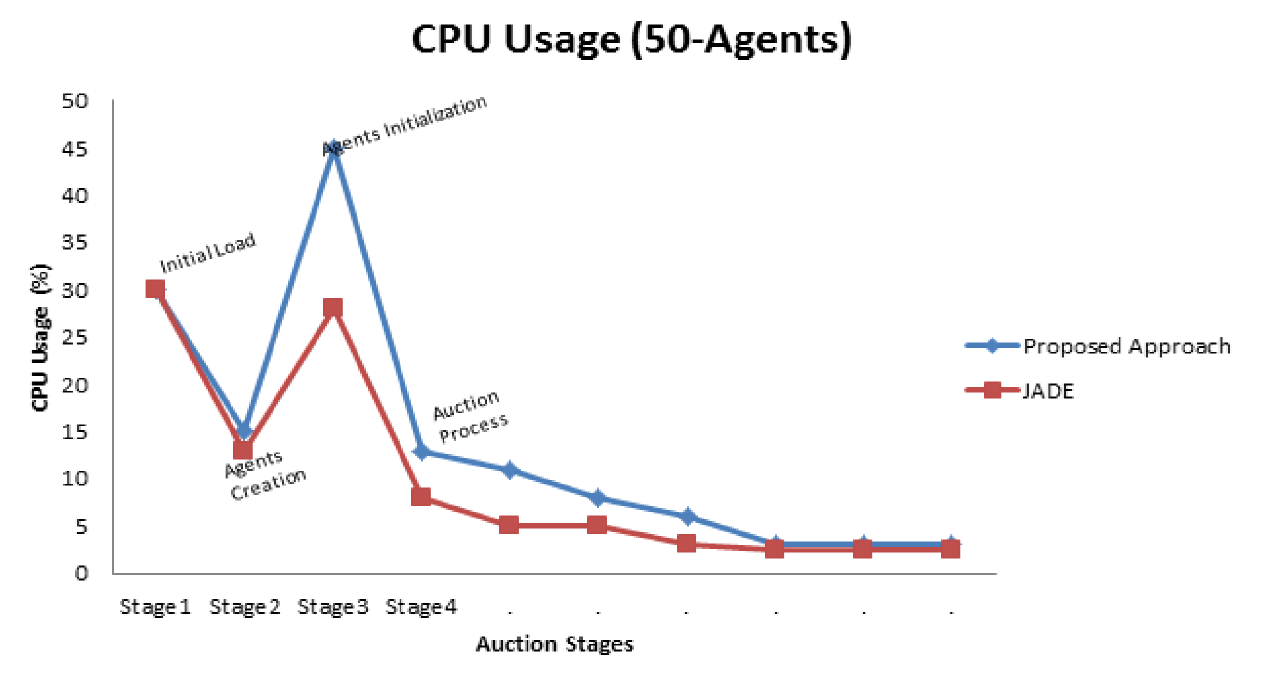
Fig 4.4Memory Utilization w.r.t Number of agents

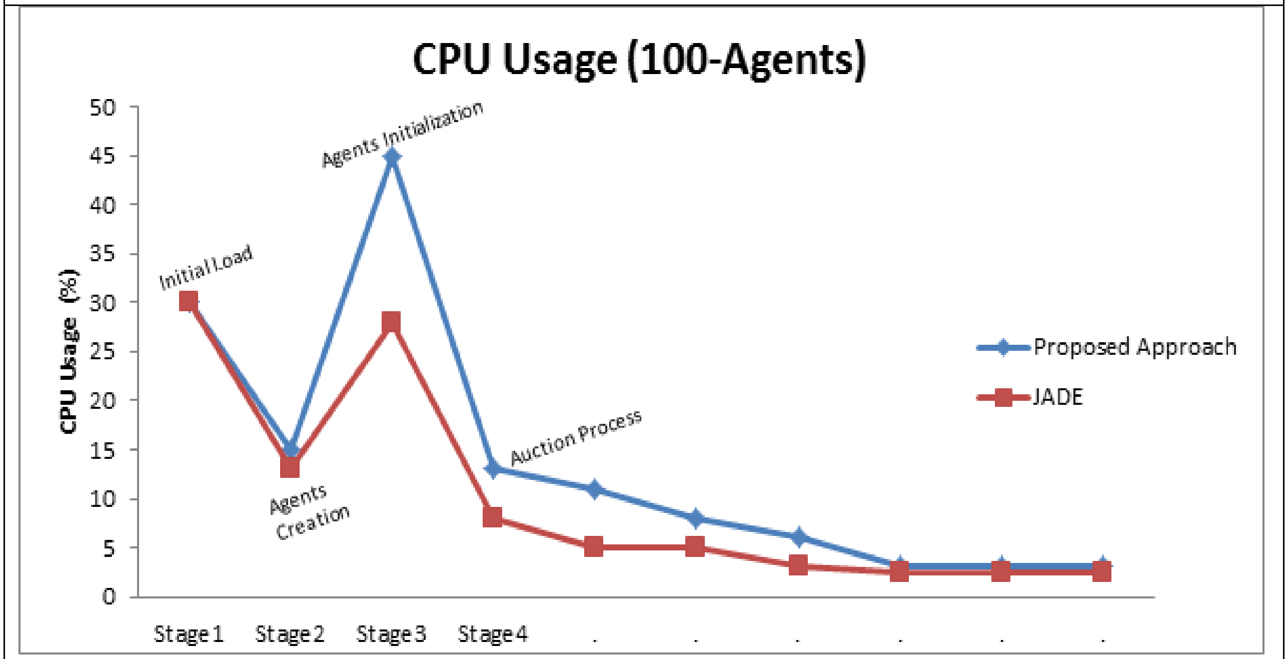
II. CPU UTILIZATION

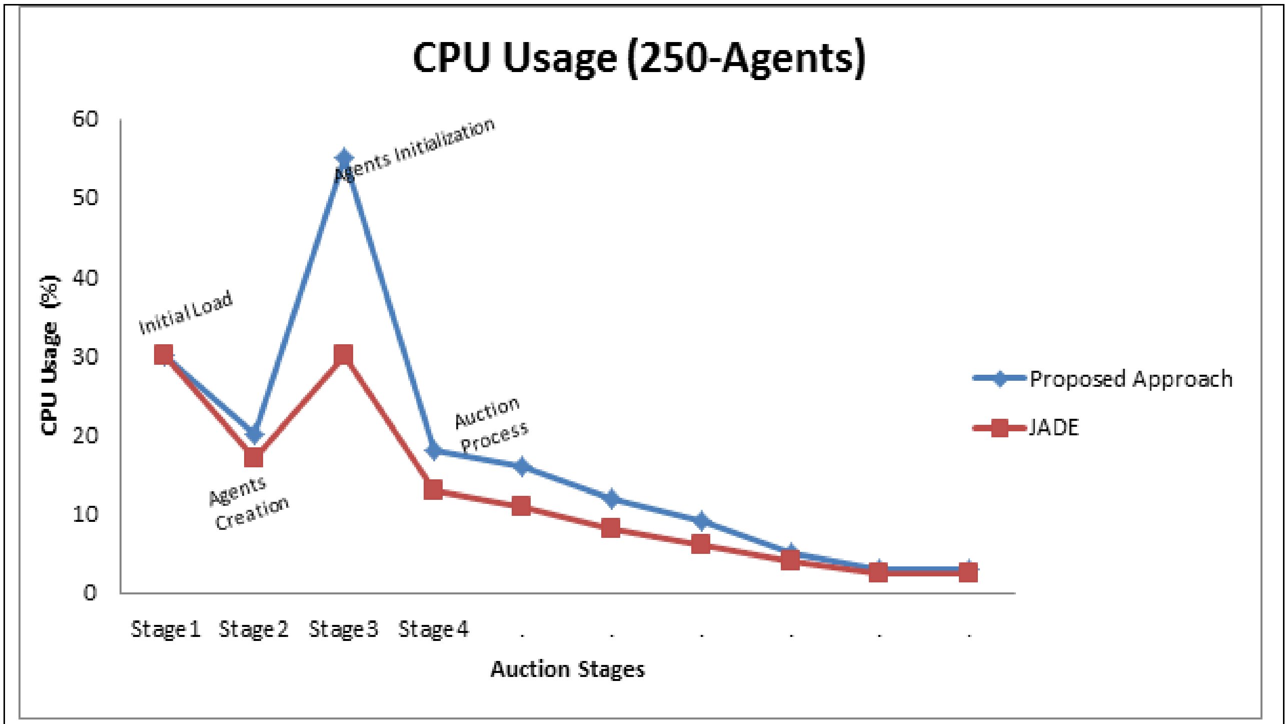
CPU utilization is a key performance metric. It can be used to track CPU performance regressions, or improvements, and is a useful source for performance problem investigations. CPU utilization has important implications on other system performance characteristics too, such as power consumption. CPU time is the overall time that all processors spent working in the application. When benchmarking the CPU utilization of the developed prototype, it was observed that the prototype developed with the proposed approach required occasionally more CPU time than the system developed using conventional JADE approach. This difference in the CPU usage was mostly at the time of agents' initialization when each agent looked into the ontology for the settings and invoked an instance of the rule engine knowledge-base. Again, it's because of the Drools rule engine which is a heavy component and utilizes most of the CPU. It is one of the main characteristic with rule engine that, the more CPU power it has the quicker it is. This growth is normal because all the Rete graph and related stuff are in RAM and Drools work on it. There is no wait for database and IO operations so CPU can go very high very quickly, but is pretty normal. On top of that, it is always good to remember that the Rete algorithm is optimized for the processing of high volumes of rules and the overall performance is not tied to the number of existing rules, but the average number of matching rules.

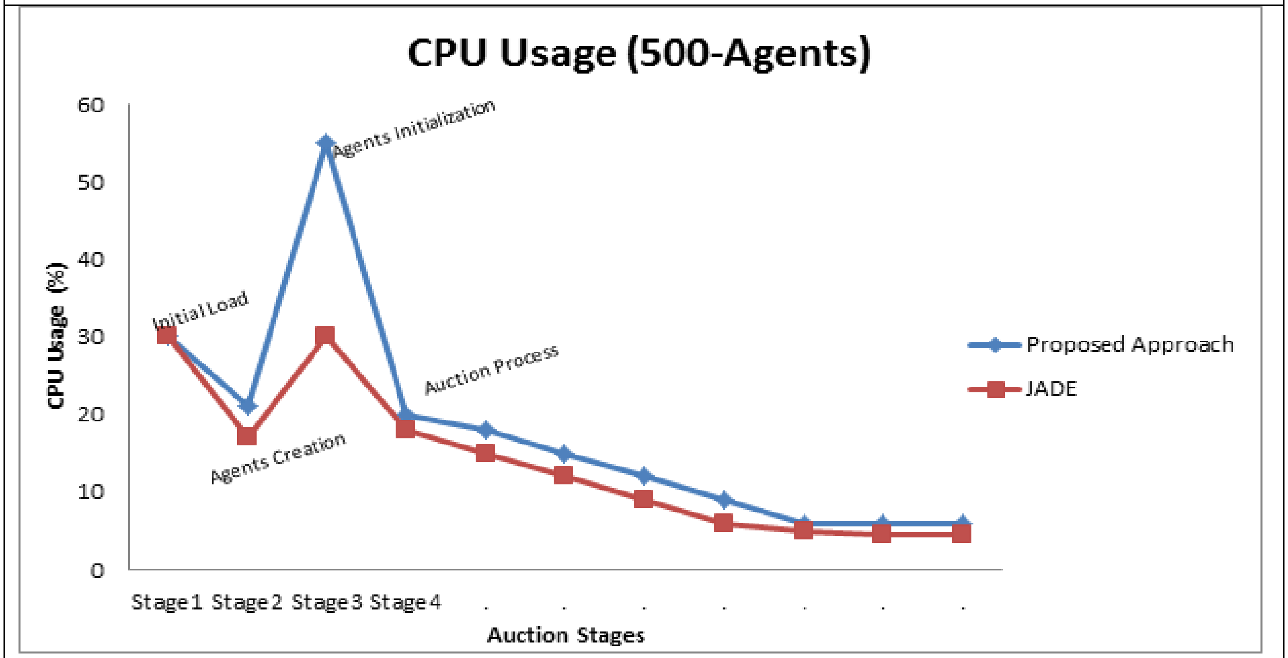
During evaluation when agents launched around 50-2000 threads of parallel execution with drools session, it performed well with more CPU usage. The system maintained average CPU usage between 15-40% with occasional spikes went to 75-80%. Since all the work is being done during fact insertion, of course adding more and more matching rules at runtime will eventually push the CPU load factor to the natural upper limit. The figures gathered during an evaluation indicated that the system has a good balance between I/O and CPU load. The increase in CPU time with increased number of matching rules is to be expected because all the work is being done during fact insertion.

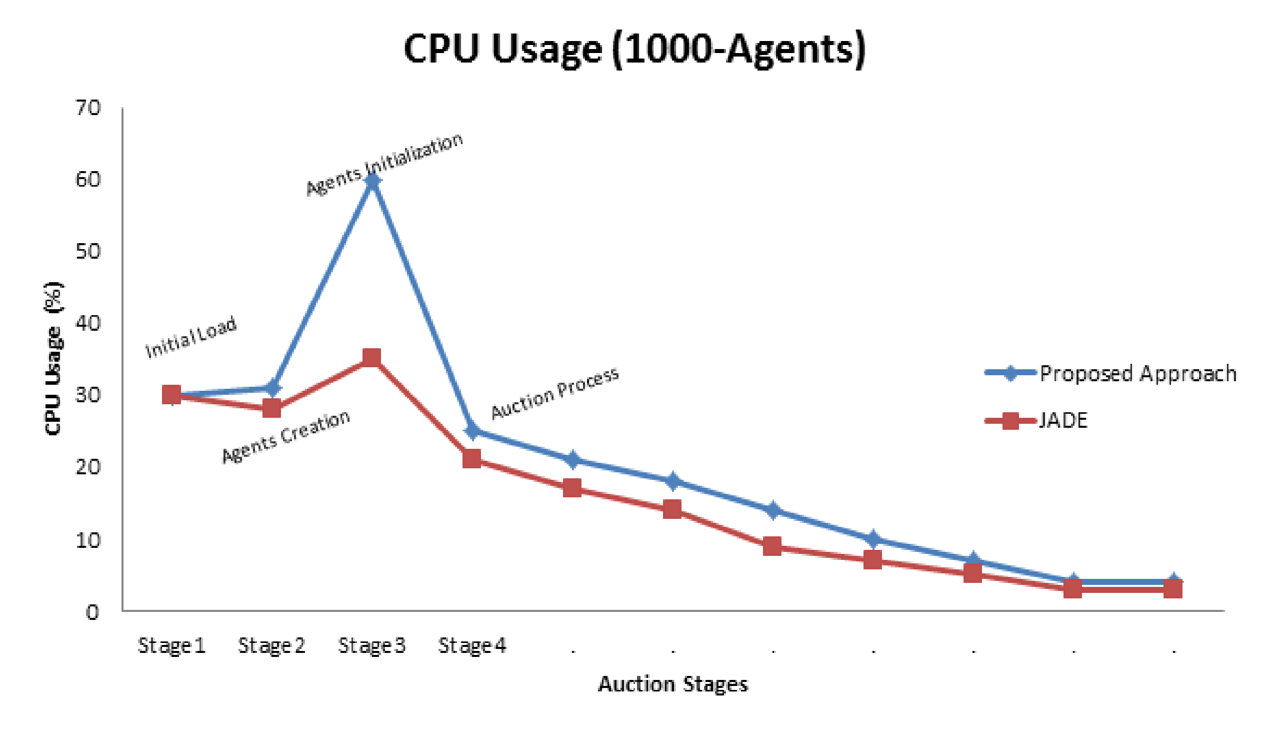
Here are the charts representing the CPU utilization of the prototype developed using the proposed model in comparison to the one developed without it. The system was tested by gradually increasing the number of agents in both prototypes.

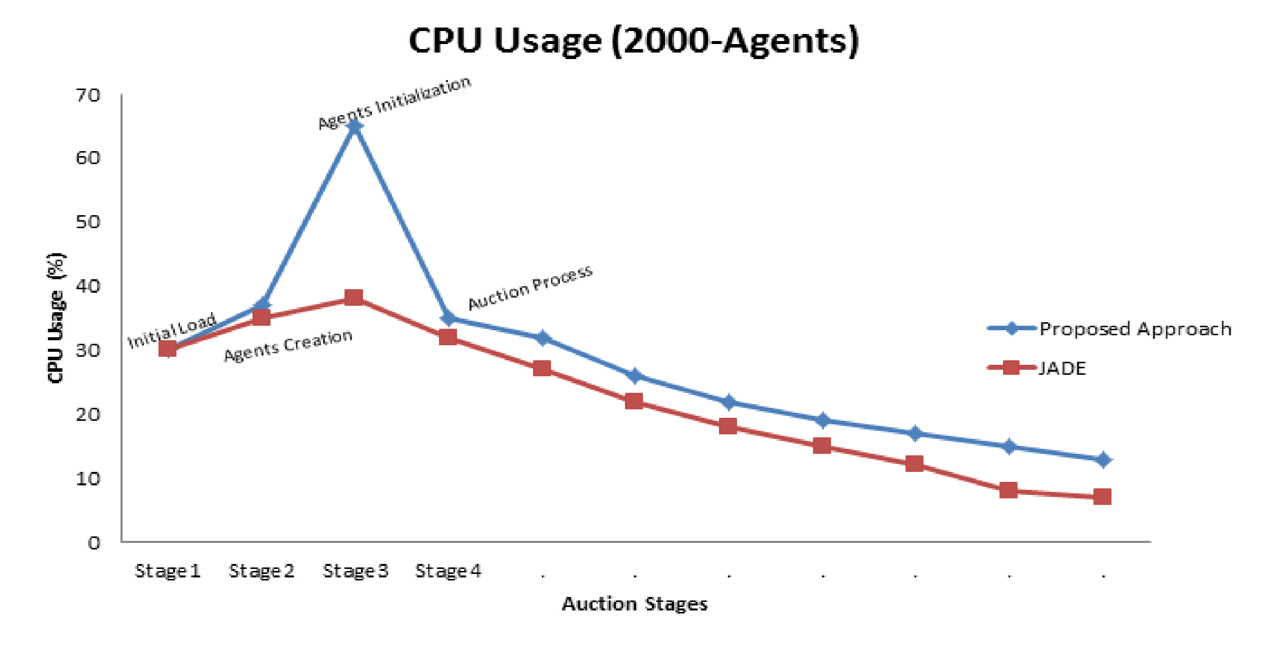

CPU Usage (50-Agents)
CPU Usage (%)
50
45
40
35
30
25
20
15
10
5
0
Initial Load
Agents Creation
Agents Initialization
Auction Process
Stage 1
Stage 2
Stage 3
Stage 4
.
.
.
.
.
.
Auction Stages
Proposed Approach
JADE


CPU Usage (100-Agents)
CPU Usage (%)
50
45
40
35
30
25
20
15
10
5
0
Initial Load
Agents Creation
Agents Initialization
Auction Process
Stage 1
Stage 2
Stage 3
Stage 4
.
.
.
.
.
.
Proposed Approach
JADE


CPU Usage (250-Agents)
CPU Usage (%)
60
50
40
30
20
10
0
Initial Load
Agents Creation
Agents Initialization
Auction Process
Stage 1
Stage 2
Stage 3
Stage 4
.
.
.
.
.
.
Auction Stages
Proposed Approach
JADE


CPU Usage (500-Agents)
CPU Usage (%)
60
50
40
30
20
10
0
Initial Load
Agents Creation
Agents Initialization
Auction Process
Stage 1
Stage 2
Stage 3
Stage 4
.
.
.
.
.
.
.
Auction Stages
Proposed Approach
JADE


CPU Usage (1000-Agents)
CPU Usage (%)
70
60
50
40
30
20
10
0
Agents Initialization
Initial Load
Agents Creation
Auction Process
Proposed Approach
JADE
Stage 1
Stage 2
Stage 3
Stage 4
.
.
.
.
.
.
.
Auction Stages

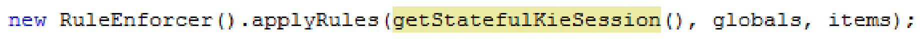

CPU Usage (2000-Agents)
CPU Usage (%)
70
60
50
40
30
20
10
0
Agents Initialization
Initial Load
Agents Creation
Auction Process
Proposed Approach
JADE
Stage 1
Stage 2
Stage 3
Stage 4
.
.
.
.
.
.
.
Auction Stages

4.3 Reconfiguration Time

There are two ways through which agents can adopt the changed behavior. One way is when the agent is created. It looks for the latest settings in the knowledge base and then gets a new knowledge session to work on. The second possible way is when the agent configuration is changed at runtime. In that way as soon as the settings in the ontology are changed, the change monitor thread captures those changes and applies those in the rule engine's knowledge-base. Whenever the agents need to perform some action, they ask for a session to work on. At that time they become aware of the changed knowledge-base, and update their knowledge accordingly. After performing the action the agents close the created session and move to the next action.

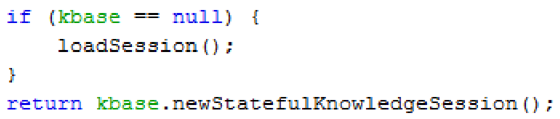
It has been analyzed that in the developed prototype, the number of agents doesn't affect the amount of configuration adaptation time. The moment the changes are made to the ontology, these are made available and agent's behaviors are reconstructed. It is successfully demonstrated in this way when during runtime the bidding behavior of agents was changed from the English auction to Dutch auction. This is because the ontology settings file is monitored for the changes and when change occurs, it is reflected in the knowledge base and each class type, e.g. Buyer, Auctioneer, etc. maintains one latest version of the knowledge base, from which all the agent instances build their sessions.

This prototype deferred the agents' adaptation to the changed rules at runtime till they needed to perform an action. This not only saves us from costly intimation of changed behavior when the number of agents is too large, but also helps to control the cost and time to test and deploy changes. When an agent has to execute a behavior, it populates the required objects, sets the global variables, and applies the rules by building a new session from the knowledge-base.



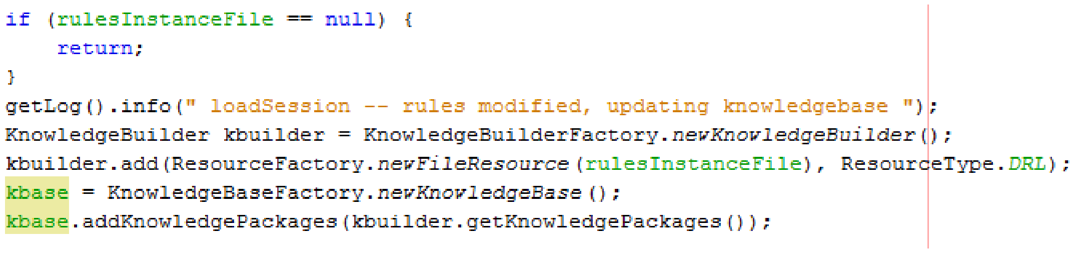
```
new RuleEnforcer().applyRules(getStatefulKieSession(), globals, items);
```

If the knowledge base is unchanged, a new session is built from it and returned to the agent.



```
if (kbase == null) {
    loadSession();
}
return kbase.newStatefulKnowledgeSession();
```

If a change is detected in the agent’s configuration, a new knowledge base is created with the updated settings and returned to the agent and the agent then build new session from this updated knowledge-base.



```
if (rulesInstanceFile == null) {
    return;
}
getLog().info(" loadSession -- rules modified, updating knowledgebase ");
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kbuilder.add(ResourceFactory.newFileResource(rulesInstanceFile), ResourceType.DRL);
kbase = KnowledgeBaseFactory.newKnowledgeBase();
kbase.addKnowledgePackages(kbuilder.getKnowledgePackages());
```

So it was demonstrated in the developed prototype that, the reconfiguration of the agents is so real that agents always get to the updated knowledge no matter how large is the number of agents.

4.4 Conclusion

The evaluation shows that system has potential to successfully change the agent’s behavior. The results show that when sufficient resources in terms of memory and CPU are available, the multi-agent system and rule-engine work efficiently. The agents look into each and every setting option before the execution of behaviors which results into the realization of most up-to-date business logic available. The memory utilization varies with the number of agents. With a smaller number of agents, there is no significant difference in the two systems, but when we gradually increase the number of agents up to 250, 500 and 1000 agents, the difference becomes almost twice. This difference tends to narrow down again as we further increase the number of agents to 2000. This can be due to the way in which the JVM scales horizontally in discrete intervals. The results demonstrate the system efficiency by adopting changes to the agent’s behaviors. Further, the results also demonstrate the suitability of an auction mechanism for real-time reconfiguration.

Chapter 5

Conclusion and Future Work

"A conclusion is the place where you got tired thinking"
Martin H. Fischer

Real-time change management is the procedure by which a system gets to its future intended vision. It has been found that the greater part of the hypotheses of software change is philosophical and has been gotten from mathematics and physics. There are two major ways in which software change are categorized.

- First order change is a variety in the way procedures and methods are done in a given software system, leaving the software itself generally unaltered. A few samples are developing new reports, making better approaches to gather the same information, and refining existing procedures and methods.

- Second order change happens when the software itself is changed. This sort of change generally happens as the consequence of a vital change or a noteworthy emergency, for example, a danger against software survival. Second order change includes a redefinition or reconceptualization of the matter of the organization and the way it is to be led. In some field, transforming from a paper based record to an electronic record is an example of a second order change, just like the ATMs at bank redefined the way that various functionalities at bank or financial organization are performed.

These changes in software are actually the changes in the functional requirements of a software system. In multi-agent systems, agent behaviors are actually revealed as these functional requirements. In this work , these behaviors are demonstrated and externalized as settings kept in the ontology. The settings are, essentially, executable requests. In the design model they are available as RDF triples. In the implementation model they are managed centrally and effortlessly changed. Since these RDF triples are suitably easy to alter, and agents always get the latest behaviors to interpret, deployment of new requirements requires negligible effort. The RDF specification of the agent-rules, identified with the relating agent behaviors, make our models which consolidate requirements with RDF reusable. The models are constantly reused for the customary correction by clients, as well as for consistent interpretation by the software agents. The upkeep of the OCMAS models is, in fact, equal to the upkeep of the final system product.

One shortcoming of OCMAS is that the model's externalization of agent behaviors as ontology settings and execution of facts in the rule-engine will affect the performance of the system. Each time there is an adjustment in the business logic, the system agent responds to change event, and updates the rules. Additionally, when the rules are executed, there is a trade-off between ease of change and performance. Despite the fact that the determination of this issue remains a part of future work, one conceivable change can be the use of solutions like the drools camel server. Drools camel server module is a war which you can deploy to execute Knowledge-Bases remotely for any sort of client application. This is not limited to JVM application clients, but any technology that can use HTTP, through a REST interface. As a result, you can set up your process engine "as a service" and integrate this into your applications easily by doing remote requests and/or sending the necessary triggers to the execution server whenever necessary (without the need to embed or manage this as part of your application).

Ultimately, we hope to accomplish real-time reconfiguration in the OCMAS where, as agents interact with the human users they improve their behaviors and intentions. This permits agents' beliefs to be updated, resultantly inferring behaviors that can be added to the central rules-record. These inferred behaviors can be shared and executed by all the agents and are subject to modifications. After sooner or later, a developed and solid rule-set, that is autonomous of those gained through the defined settings, can be built up.

OCMAS would be valuable for those spaces that have frequently changing business requirements where redevelopment would somehow be expensive. Especially, OCMAS ought to function fine when collaboration is required between various elements and where this cooperation may be liable for changes, as an aftereffect of changing business logic. OCMAS is likewise suitable where the business environment is oftentimes changing with developing ideas and behaviors. The future work will incorporate the richer and more descriptive ontologies to handle even complex business rules. The Ontology centric multi-agent framework model will be made all the more capable and more adaptable, however the work so far shows that it is profoundly significantly and valuable to the evolutionary and developmental needs of multi-agent framework systems.

References

[1] Deen,S.M, Ponnamperuma, K. "Dynamic ontology integration in a multi-agent environment" Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference.

[2] Quynh-Nhu Numi Tran , Graham Low "MOBMAS: A methodology for ontology-based multi-agent systems development" 14 Jan 2007, http://www.sciencedirect.com/science/article/pii/S0950584907000766

[3] Chen,S.C. Dow, C.R. ; Yang, T.K. ; Bai, J.Y. ; Lin, C.M. "An instant messaging-based multi-agent coordination system" Information Reuse and Integration, 2004. IRI 2004. Proceedings of the 2004 IEEE International Conference,

[4] DiLeo, Jonathan; Jacobs, Timothy ; DeLoach, Scott "Integrating Ontologies into Multiagent Systems Engineering", 2006, http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA44370

[5] Warwas, S. "Making Multiagent System Designs Reusable: A Model-Driven Approach" Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011 IEEE/WIC/ACM International Conference, pages: 22-27 Aug. 2011.

[6] Peter Bloodsworth, Sue Greenwood, "COSMOA an ontology-centric multi-agent system for co-ordinating medical responses to large-scale disasters", IOS Press 2005.

[7] D .Fensel, S .Decker, M .Erdmann and R .Studer, “Ontobroker in a Nut-shell”, Lecture Notes in Computer Science, Issue 1513, pages: 663-664, 1998.

[8] Pekka Isto and Jarmo Korhonen, “Practical Experiences in Developing Ontology-Based Multi-Agent System”, Proceedings of Business Information Systems (BIS 2003), 2003.

[9] I .Blacoe, I .Dickinson, V .Tamma and M .Wooldridge, “An Ontology Based Approach to Automated Negotiation”, Lecture Notes in Computer Science, Issue 2531, pages: 219-237, 2002.

[10] S.F .Smith and M.A .Becker, “An Ontology for Constructing Scheduling Systems”, Technical Report- American Association for Artificial Intelligence, Number 06-97, pages: 120-129, 1997.

[11] Wang B. and Liu L, “Ontology-Based Multi-Agent Diagnostic System of Enterprise Management”, IEEE International Conference on Computer Science and Automation Engineering (CSAE), pages: 577-581, 2012.

[12] Korhonen, J., Isto, P, “Practical Experiences in Developing Ontology-Based MultiAgent System”, Proceedings of the 6th International Conference on Business Information Systems (BIS 2003) Colorado Springs, USA, pages: 147-152, 2003.

[13] Munir Merdan, Lisa Vittori, Gottfried Koppensteiner, Pavel Vrba, Bernard Favre-Bulle, "Simulation of an ontology-based multi-agent transport system", SICE Annual Conference, The University Electro-Communications, 2008.

[14] Gruber T. R. "A translation approach to portable Ontologies Knowledge Acquisition", pages: 199-220, 1993.

[15] Merdan M.; Koppensteiner G.; Hegny I. & Favre-Bulle B, “Application of an Ontology in a Transport Domain”, IEEE International Conference on Industrial Technology (IEEE ICIT2008), 2008, Sichuan University, Chengdu, China

[16] S.F. Smith and M. Becker, “An Ontology for Constructing Scheduling Systems,” presented at AAAI Spring Symposium on Ontological Engineering, Stanford University, 1997.

[17] Valentina Tamma, Michael Wooldridge, and Ian Dickinson, "An ontology based approach to automated negotiation". In Proceeding of Agent-Mediated Electronic Commerce (AMEC’IV) workshop at AAMAS’02, 2002.

[18] Andrew Branson; Richard McClatchey: “CRISTAL A practical study in designing systems to cope with change”, Information Systems 42 (2014), pp 139-152 Elsevier Publishers

[19] KLICK-KLOCK, Klick-Klock Productions LLC ”Conducting interactive auctions with prices continually adjusted over a fixed period analogous to that of a dutch auction, online, via a global computer network”, <http://www.trademarkia.com/klikklok-75302094.html>

[20] Timeshare Resales International, “Timeshare Resales International,” http://hoteltimeshareresales.net/, June, 2014.

[21] Antebellum Covers, “Antebellum Covers,” http://www.antebellumcovers.com/, Mar, 2014.

[22] ebay, “Electronics, Car, Fashion, Collectibles, Coupons and More Online Shopping,” http://www.ebay.com/, June, 2015.

[23] Rete Algorithm, “Rete Algorithm – Wikipedia, the free encyclopedia,” https://en.wikipedia.org/?title=Rete_algorithm, Mar, 2015.

Appendices

Appendix A

Agent Functions and Processes Settings Represented in XML.

```
<functions>
    <buyer>
        <function name="Bid To Auction">
            <state value="ON" />
            <processes>
                <process name="Log" state="ON" />
            </processes>
        </function>
        <function name="Make Payment">
            <state value="ON" />
            <processes>
                <process name="Log" state="ON" />
            </processes>
        </function>
        <function name="Reject Offer">
            <state value="OFF" />
            <processes>
                <process name="Log" state="ON" />
            </processes>
        </function>
        <function name="Accept Offer">
            <state value="OFF" />
            <processes>
                <process name="Log" state="ON" />
            </processes>
        </function>
    </buyer>
    <auctioneer>
```

```
<functions>
    <buyer>
        <function name="Bid To Auction">
            <state value="OFF" />
            <processes>
                <process name="Log" state="ON" />
            </processes>
        </function>
        <function name="Make Payment">
            <state value="ON" />
            <processes>
                <process name="Log" state="ON" />
            </processes>
        </function>
        <function name="Reject Offer">
            <state value="ON" />
            <processes>
                <process name="Log" state="ON" />
            </processes>
        </function>
        <function name="Accept Offer">
            <state value="ON" />
                        <processes>
                <process name="Log" state="ON" />
            </processes>
        </function>
    </buyer>
    <auctioneer>
```

<function name="Enable highest bid broadcast">	<function name="Enable highest bid broadcast">
<state value="ON" />	<state value="OFF" />
</function>	</function>
<function name="Broadcast Auction">	<function name="Broadcast Auction">
<state value="ON" />	<state value="OFF" />
<processes>	<processes>
<process name="Log" state="ON" />	<process name="Log" state="ON" />
</processes>	</processes>
</function>	</function>
<function name="Accept Bid">	<function name="Accept Bid">
<state value="ON" />	<state value="OFF" />
<processes>	<processes>
<process name="Log" state="ON" />	<process name="Log" state="ON" />
</processes>	</processes>
</function>	</function>
<function name="Reject Bid">	<function name="Reject Bid">
<state value="ON" />	<state value="OFF" />
<processes>	<processes>
<process name="Log" state="ON" />	<process name="Log" state="ON" />
</processes>	</processes>
</function>	</function>
<function name="Award Winner">	<function name="Award Winner">
<state value="ON" />	<state value="OFF" />
<processes>	<processes>
<process name="Log" state="ON" />	<process name="Log" state="ON" />
<process name="Send Email" state="ON" />	<process name="Send Email" state="ON" />
</processes>	</processes>
</function>	</function>
<function name="Authorize Shipping">	<function name="Authorize Shipping">
<state value="ON" />	<state value="ON" />
<processes>	<processes>
<process name="Log" state="ON" />	<process name="Log" state="ON" />

<process name="Send Email" state="ON" />	<process name="Send Email" state="ON" />
</processes>	</processes>
</function>	</function>
<function name="Broadcast Proposal">	<function name="Broadcast Proposal">
<state value="OFF" />	<state value="ON" />
<processes>	<processes>
<process name="Log" state="ON" />	<process name="Log" state="ON" />
</processes>	</processes>
</function>	</function>
<function name="Offer Accepted">	<function name="Offer Accepted">
<state value="OFF" />	<state value="ON" />
<processes>	<processes>
<process name="Log" state="ON" />	<process name="Log" state="ON" />
<process name="Send Email" state="ON" />	<process name="Send Email" state="ON" />
</processes>	</processes>
</function>	</function>
<function name="Offer Rejected">	<function name="Offer Rejected">
<state value="OFF" />	<state value="ON" />
<processes>	<processes>
<process name="Log" state="ON" />	<process name="Log" state="ON" />
</processes>	</processes>
</function>	</function>
</auctioneer>	</auctioneer>
</functions>	</functions>

Appendix B

Drools Rules Represented in .drl file.

DroolsAuctioneer.drl

```
dialect "mvel"
global com.agent.auction.jade.drools.DroolsAuctioneer auctioneer
import com.agent.auction.jade.drools.DroolsAuction
import java.lang.Double


rule "Configure"
   when
      /*control*/eval(true)
   then
      auctioneer.setReturnHigestBid(false)
end
rule "Enable highest bid broadcast"
   when
      /*control*/eval(true)
   then
      auctioneer.setReturnHigestBid(true)
end
rule "Broadcast Auction"
when
   $a: DroolsAuction(leadingBuyer==null,expired==false)
   /*control*/eval(true)
then
   /*process*/auctioneer.logRule("Broadcast Auction")
   $a.setProposing(false)
   auctioneer.broadcastAuction($a)
end

rule "Accept Bid"
when
   $current: Double()
   $a: DroolsAuction(proposing==true,leadingBuyer!=null,expired==false)
   eval($a.proposedPriceHigherThanCurrent($current))
   /*control*/eval(true)
then
   /*process*/auctioneer.logRule("Accept Bid")
   $a.setAccepted(true)
   auctioneer.notifyCurrentLeading($a)
   auctioneer.respondToBid($a)
end
rule "Reject Bid"
when
   $current: Double()
```

```
    $a: DroolsAuction(proposing==true,leadingBuyer!=null,expired==false)
    not (eval($a.proposedPriceHigherThanCurrent($current)))
    /*control*/eval(true)
then
    /*process*/auctioneer.logRule("Reject Bid")
    $a.setAccepted(false)
    auctioneer.respondToBid($a)
end
rule "Award Winner"
when
    $a: DroolsAuction(proposing==true,expired==true,leadingBuyer!=null,won==false)
    /*control*/eval(true)
then
    /*process*/auctioneer.logRule("Award Winner")
    /*process*/auctioneer.requestEmail($a.getLeadingBuyer(),"You have won the auction
"+$a.getId(), $a.getWinningMailable())
    auctioneer.won($a)
end
rule "Authorize Shipping"
    when
        $bn: DroolsAuction(paid==true)
         /*control*/eval(true)
    then
        /*process*/auctioneer.logRule("Authorize Shipping")
        /*process*/auctioneer.requestEmail($bn.getLeadingBuyer(), "Payment Confirmed",
$bn.getPaymentConfirmedMailable())
        auctioneer.authorizeShipping($bn)
end
rule "Broadcast Proposal"
when
    $a: DroolsAuction(leadingBuyer==null)
    /*control*/eval(false)
then
    /*process*/auctioneer.logRule("Broadcast Proposal")
    $a.setProposing(true)
    auctioneer.broadcastAuction($a)
end
rule "Offer Accepted"
when
    $a: DroolsAuction(leadingBuyer!=null,accepted==true,proposing==false,won==false)
    /*control*/eval(false)
then
    /*process*/auctioneer.logRule("Offer Accepted")
    $a.setProposing(true)
    /*process*/auctioneer.requestEmail($a.getLeadingBuyer(),"You have won the auction
"+$a.getId(), $a.getWinningMailable())
    auctioneer.won($a)
end
```

```
rule "Offer Rejected"
when
   $a: DroolsAuction(leadingBuyer!=null,accepted==false,proposing==false,won==false)
   /*control*/eval(false)
then
   /*process*/auctioneer.logRule("Offer Rejected")
   $a.lowerPriceBy(2,10)
   auctioneer.reviseProposal($a)
end
```

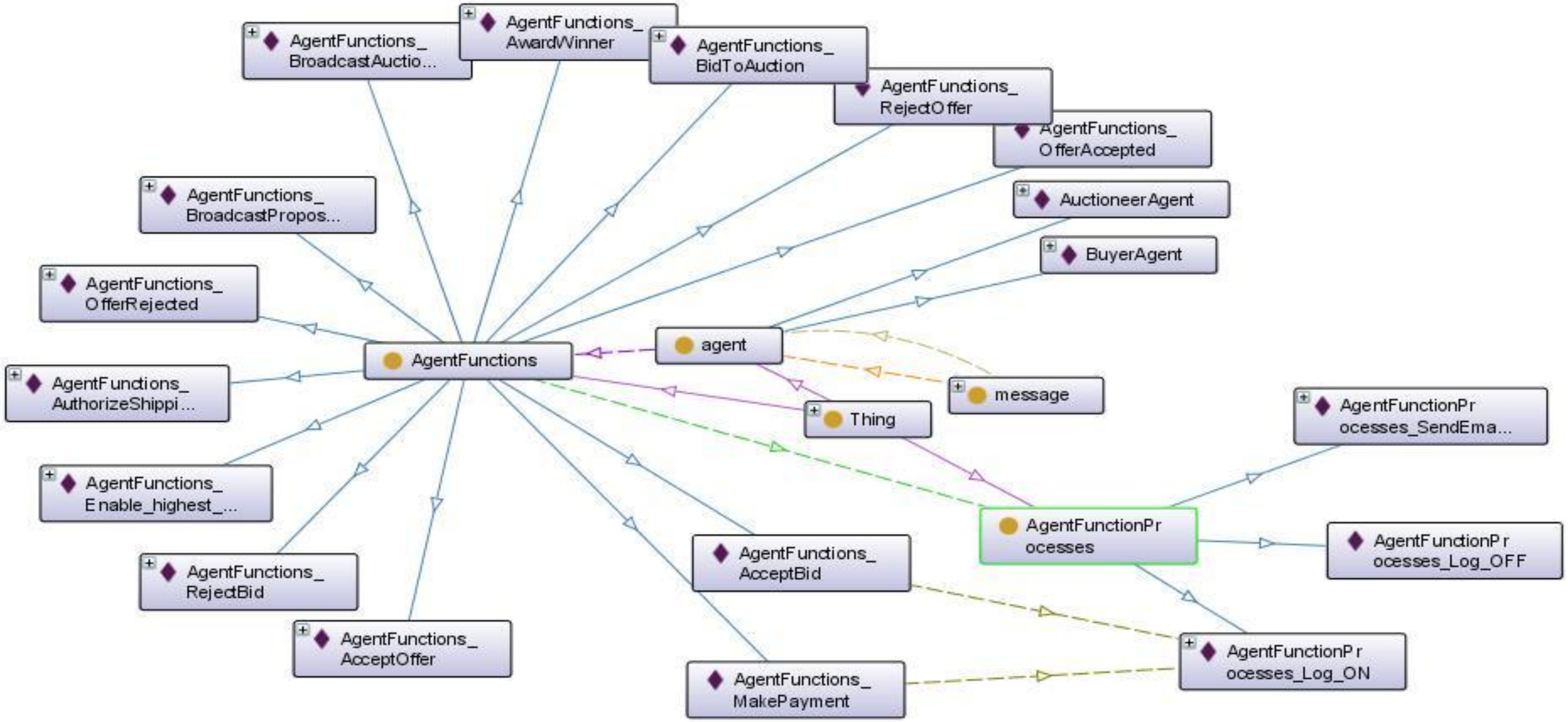
DroolsBuyer.drl

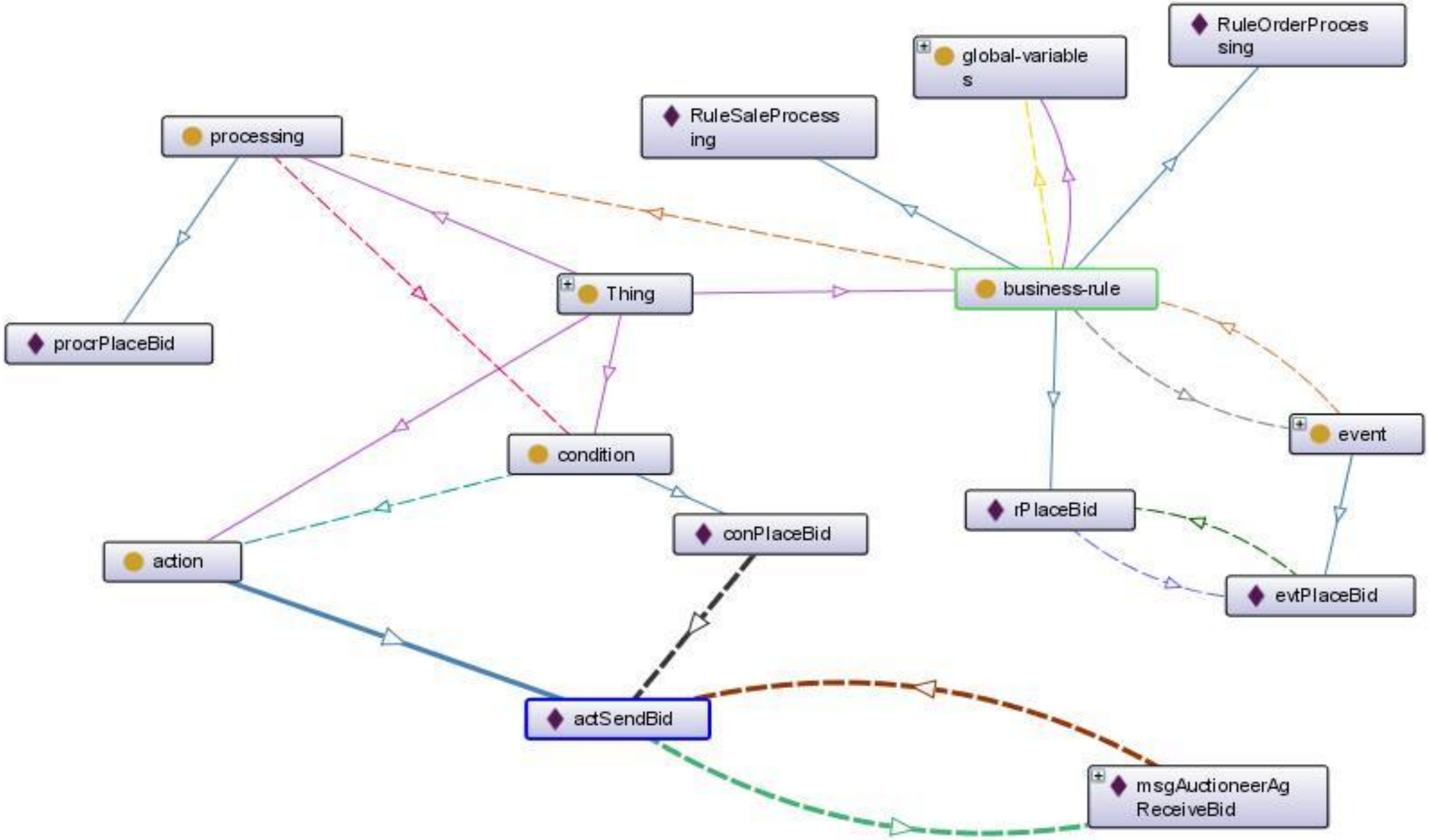
```
dialect "mvel"
global com.agent.auction.jade.drools.DroolsBuyer buyer
import com.agent.auction.jade.drools.DroolsAuction
/**
 *
 * @author mjan
 */

rule "Configure"
 when
   eval(true)
 then
   /*process*/buyer.logRule("Configure")
   buyer.setMinimumDiscountPercentage(15,30)
   buyer.setResponseDelay(5,20)
 end
rule "Bid To Auction"
when
   $a: DroolsAuction(proposing==false,accepted==false, won==false)
   /*control*/eval(true)
then
   /*process*/buyer.logRule("Bid To Auction")
   $a.raisePriceBy(1,7)
   buyer.bidToAuction($a)
end

rule "Make Payment"
when
   $a: DroolsAuction(paid==true)
   /*control*/eval(true)
then
   /*process*/buyer.logRule("Make Payment")
   buyer.pay($a)
end
rule "Reject Offer"
```

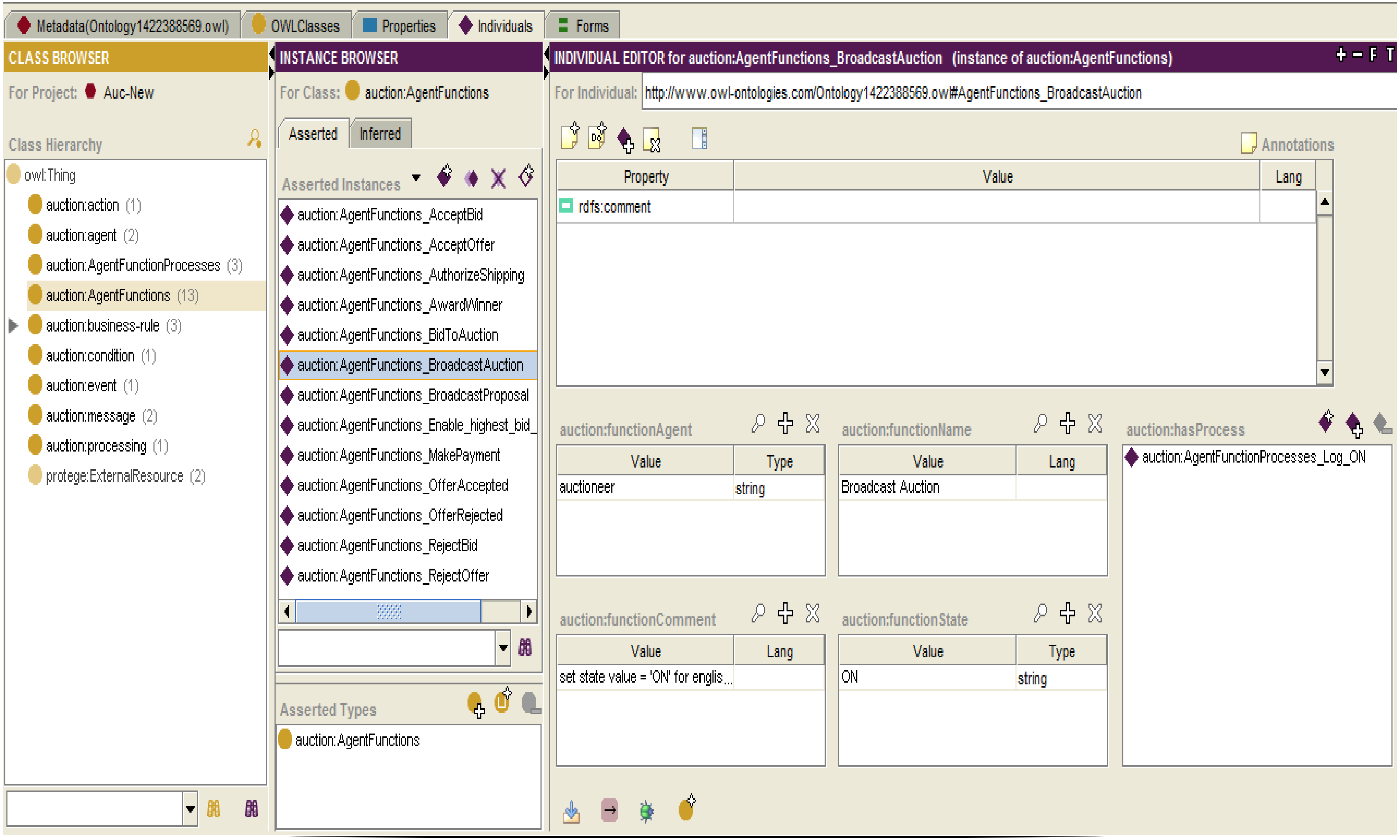
```
 when
   $a: DroolsAuction(proposing==true)
   not
(eval($a.proposedPriceLowerStartingBy(buyer.getMinimumDiscount(),buyer.getMinimumDis
count())))
   /*control*/eval(false)
 then
   /*process*/buyer.logRule("Reject Offer")
   $a.setAccepted(false)
   buyer.respondToProposal($a)
end
 rule "Accept Offer"
 when
   $a: DroolsAuction(proposing==true)

eval($a.proposedPriceLowerStartingBy(buyer.getMinimumDiscount(),buyer.getMinimumDisc
ount()))
   /*control*/eval(false)
 then
   /*process*/buyer.logRule("Accept Offer")
   $a.setAccepted(true)
   buyer.respondToProposal($a)
end
```

APPENDIX C

ONTOLOGY TO STRUCTURE AGENT BEHAVIORS (TREE GRAPHS).


AgentFunctions_ BroadcastAuctio...
AgentFunctions_ AwardWinner
AgentFunctions_ BidToAuction
AgentFunctions_ RejectOffer
AgentFunctions_ OfferAccepted
AuctioneerAgent
BuyerAgent
AgentFunctions_ BroadcastPropos...
AgentFunctions_ OfferRejected
AgentFunctions
agent
message
Thing
AgentFunctions_ AuthorizeShippi...
AgentFunctionPr ocesses_SendEma...
AgentFunctions_ Enable_highest_...
AgentFunctionPr ocesses
AgentFunctionPr ocesses_Log_OFF
AgentFunctions_ AcceptBid
AgentFunctions_ RejectBid
AgentFunctions_ AcceptOffer
AgentFunctionPr ocesses_Log_ON
AgentFunctions_ MakePayment


RuleOrderProcessing
global-variables
RuleSaleProcessing
processing
Thing
business-rule
procrPlaceBid
condition
event
rPlaceBid
conPlaceBid
action
evtPlaceBid
actSendBid
msgAuctioneerAgReceiveBid



Metadata(Ontology1422388569.owl) | OWLClasses | Properties | Individuals | Forms

CLASS BROWSER

For Project: Auc-New

Class Hierarchy

- owl:Thing
  - auction:action (1)
  - auction:agent (2)
  - auction:AgentFunctionProcesses (3)
  - auction:AgentFunctions (13)
  - auction:business-rule (3)
  - auction:condition (1)
  - auction:event (1)
  - auction:message (2)
  - auction:processing (1)
  - protege:ExternalResource (2)

INSTANCE BROWSER

For Class: auction:AgentFunctions

Asserted | Inferred

Asserted Instances

- auction:AgentFunctions_AcceptBid
- auction:AgentFunctions_AcceptOffer
- auction:AgentFunctions_AuthorizeShipping
- auction:AgentFunctions_AwardWinner
- auction:AgentFunctions_BidToAuction
- auction:AgentFunctions_BroadcastAuction
- auction:AgentFunctions_BroadcastProposal
- auction:AgentFunctions_Enable_highest_bid_
- auction:AgentFunctions_MakePayment
- auction:AgentFunctions_OfferAccepted
- auction:AgentFunctions_OfferRejected
- auction:AgentFunctions_RejectBid
- auction:AgentFunctions_RejectOffer

Asserted Types

- auction:AgentFunctions

INDIVIDUAL EDITOR for auction:AgentFunctions_BroadcastAuction (instance of auction:AgentFunctions)

For Individual: http://www.owl-ontologies.com/Ontology1422388569.owl#AgentFunctions_BroadcastAuction

Annotations

Property	Value	Lang
rdfs:comment		

auction:functionAgent

Value	Type
auctioneer	string

auction:functionName

Value	Lang
Broadcast Auction	

auction:hasProcess

- auction:AgentFunctionProcesses_Log_ON

auction:functionComment

Value	Lang
set state value = 'ON' for englis...	

auction:functionState

Value	Type
ON	string