AGENT-MEDIATED BROKERING AND MATCHMAKING FOR

SIMULATION MODEL REUSE ON THE SEMANTIC GRID

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

_____

Swetha Paspuleti

Certificate of Approval:

_____	_____
David Umphress Associate Professor Computer Science and Software Engineering	Levent Yilmaz, Chair Assistant Professor Computer Science and Software Engineering
_____	_____
Sanjeev Baskiyar Assistant Professor Computer Science and Software Engineering	Stephen L. McFarland Dean Graduate School

AGENT-MEDIATED BROKERING AND MATCHMAKING FOR SIMULATION MODEL REUSE ON THE SEMANTIC GRID

Swetha Paspuleti

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
December 16, 2005

AGENT-MEDIATED BROKERING AND MATCHMAKING FOR

SIMULATION MODEL REUSE ON THE SEMANTIC GRID

Swetha Paspuleti

_____

Signature of Author

_____

Date of Graduation

VITA

Swetha Paspuleti, daughter of P. Venkat Reddy and P. Jayapradha, was born in Nalgonda, Andhra Pradesh, India. She graduated from St. Ann's High School, Tarnaka, Hyderabad in 1997. She attended Little Flower Junior College, Hyderabad, for two years. She earned the degree Bachelor of Engineering in Computer Science Engineering from Osmania University, Hyderabad, India in 2003.

THESIS ABSTRACT

AGENT-MEDIATED BROKERING AND MATCHMAKING FOR

SIMULATION MODEL REUSE ON THE SEMANTIC GRID

Swetha Paspuleti

Master of Science, December 16, 2005
(B.E., Osmania University, Hyderabad, India, 2003)

94 Typed Pages

Directed by Levent Yilmaz

The need for simulation in military is increasing significantly due to high costs and safety limitations. Simulation in military is employed for training individuals in real-operating scenarios. The virtual, simulated models result in a synthetic environment that accounts for real-world environment and tremendously reduces costs. These simulated models are available for users through several repositories.

This thesis introduces an Agent-Mediated Brokering and Matchmaking for Simulation Model Reuse on the Semantic Grid (ABMS). ABMS facilitates discovery, location and retrieval of simulation models. The ABMS is a client/server application and consumers and service providers, who develop and publish the models, interact with the system.

It is built on the grid that provides basic discovery, notification and subscription facilities. Intelligent agents act on the grid to perform brokering and matchmaking schemes. These agents interpret and filter the metamodels associated with models to compute the degree of similarity of the identified models to the intended or required model and present various brokering modes to facilitate location transparency between the consumers and the model providers.

ACKNOWLEDGEMENTS

I would like to express my humble gratitude to Dr. Levent Yilmaz, my advisor and committee chair, for his support, patience and insightful guidance throughout my research work and graduate study. I would like to thank my committee members for taking the time out of their schedules to be on my committee. I am indebted to my parents and brother for their tremendous support, love and encouragement. I am grateful to all my friends for being there and providing emotional and moral support. Finally, Thanks to all the anonymous people who have helped me make this thesis possible.

Style manual or journal used APA Style

Computer software used Microsoft Word. Images drawn using Microsoft PowerPoint, UML Diagrams drawn using Microsoft Visio and Visual Paradigm.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The use of military simulation has increased significantly through the years with the budget and safety limitations associated in training individuals in real-operating scenarios [CAE 2004; Liophant 2003]. Due to the lack of availability of sufficient equipment for training all individuals, high costs associated with them and safety and security constraints associated with various training sessions, there exists significant focus on simulating various control systems for various kinds of training and preparation [Page and Smith 1998; OneSAF 1999]. These integrated virtual, real-like simulations present a synthetic environment that accounts for real-world environment and would tremendously reduce costs.

As simulation models are constructed, they need to be available to facilitate their accessibility from various locations. As the collection of simulation models increase so does the difficulty and complexity involved in discovering, locating and reusing complex domain specific models. Although conventional repositories and their retrieval mechanisms would satisfy the purpose, difficulties arise for the model providers or model users in having a preliminary knowledge of the existence of the repositories and their structure to arduously locate model resources [Zeigler 2000; Tolk 2004]. In addition,

as the number of potential model developers increase, discovery and reuse of these new models will become extremely difficult.

Conventional methods of retrieving information about models are passive. That is, they are consumer driven: - users should locate the model sites and have to retrieve information from the model sites. New model developers have no means of informing the users about their presence and thus promoting their products and services. In addition they have no means of making notifications to the users regarding any modifications and updates to the existing models [Kuokka and Harada 1995]. The users need to constantly interact with the system to browse and retrieve newly published models. Instead the users would be more interested in being informed when the desired model is available in the future. A system that would notify the arrival of new models would be a significant advancement in the discovery and retrieval of models over the existing simulation model repositories. In addition, the conventional methods of discovering models have no means of retrieving "sufficiently similar" or partially matched models if the exactly matched models are not available. A "sufficiently similar" model is one which has some degree of similarity to the requested model if not 100% similarity [Paolucci et al. 2002]. Protocols that can perform such flexible matches are missing in the existing repositories developed for model discovery and retrieval.

Existing discovery and retrieval methods lack these features of facilitating recommendations, subscriptions/notifications and partial degree matches. The objective of this thesis is to work on how to build an infrastructure over the Grid [Foster and Kesselman 1999; Foster et al. 2001; Foster et al. 2002] that can provide the above desired

capabilities. The goal is to determine the technologies and protocols that can form a strong foundation in providing such capabilities.

In building the desired infrastructure, agents [Sycara et al 2002; Genesereth and Ketchpel 1994; Hyacinth 1996], grid services [Foster et al. 2002; IBM Redbooks 2004] and brokering/matchmaking protocols are studied. Agents are intelligent computer programs that act autonomously on behalf of their users, across open and distributed environments, to solve a growing number of complex problems. Agents are generally responsible for participating in the system and exchanging and retrieving information for the user's reference [Retsina 2001]. There are several types of agents like user agents [Hyacinth 1996], middle agents [Sycara et al 2002], task agents [Retsina 2001], and information agents [Leonard 1993]. The concept of user agents, information agents and middle agents are used in this work.

Many technologies related to building the client/server application have been studied. The latest in this area is the Grid Services, an extension of web services, Web services have several advantages over other technologies like RMI [Ken 1998], CORBA [Ken 1998] as they are platform independent and language independent and they use HTML for transmitting messages. But it has disadvantages of overhead and lack of versatility in providing services such as persistency, notification and lifecycle management. These disadvantages are taken care of in Grid Services that provides lifecycle management, notifications, is stateless, non-transient and has several other advantages [Borja Sotomayor 2003]. While Grid services are highly promoted for resource sharing [Foster et al. 2001; Foster et al. 2002], existing infrastructure does not

provide semantic matchmaking and various alternative-brokering mechanisms discussed here. The dynamic matchmaking, discovery of models and their brokering can be deployed over the facilities provided by grid services like registry, service discovery, subscription and notification services.

In providing a recommendation of sufficient or partially matched models many protocols in different scenarios have been studied [Durfee et al 1997; Kuokka and Harada 1995; Kawamura et al 2002; Sycara et al. 2002]. Matchmaking is the cooperative partnership between information providers and consumers along with the help of an intelligent facilitator [Genesereth 1992]. Using this approach information provider actively involves in publishing their capabilities and services to the matchmaker and the consumers send requests for their desired information to the matchmaker. Matchmaking enables the providers and requesters to exchange dynamically changing information in a more effective and active manner than the traditional methods. Matchmaking mechanisms have been widely used in various applications and fields, where information changes rapidly, like product development and crisis management [Kuokka and Harada 1995]. The approach used in this thesis has the following advantages. (1) By deploying brokering protocols using agents on the grid, the advantages of brokering protocols like communication and location transparency will improve the Grid infrastructure and (2) improved matchmaking mechanisms using concept similarity, attribute similarity (linear metrics), and structure similarity (structure metric) will provide flexible, efficient and effective results than conventional key word matching mechanism.

The proposed infrastructure is built as a layered architecture. The lowest level layer is the grid architecture providing the basic services of our infrastructure such as model discovery, subscription and notification. These services deal with the publishing and the retrieval of the models. The next layer forms the intelligent services layer, which provides agent based brokering and matchmaking capabilities that are responsible for facilitating the qualification of "sufficiently similar" matched models and presenting them to the user. It's the primary responsibility of these agents (1) to filter the metamodels associated with models to compute the degree of similarity of the identified models to the intended or required model and (2) to provide various brokering modes. The infrastructure is comprised of several modules:

- Advertisement database is responsible for the storage of information related to models for facilitating matchmaking.
- Matchmaking Engine is responsible for matching between the requests and advertisements by performing two-phase filtering of matched models.
- Consumer or customer agent interacts with users and obtains inputs with respect to desired model characteristics.
- Producer or model publisher agent interacts with model providers and obtains inputs to facilitate publishing models.
- Various protocols such as recommendation, recruitment and subscription/notification facilitate brokering among producers and consumers.

The grid based active model retrieval mechanism along with the used agent technologies facilitates an environment where model developers and users need not be

aware of existence of various repositories. Furthermore, users have the advantage of being notified when the desired model is available. In addition, collaboration over the grid architecture along with intelligent matchmaking and brokering services can be used in similar projects in e-business and e-services like distributed virtual workgroups such as multi-vendor design teams and virtual corporations, where several businesses and partners interact with each other for information sharing and retrieval.

The rest of the thesis is organized as follows. Chapter 2 introduces the related work in Matchmaking, Grid architecture, Agents and Brokering protocols and the pros and cons of existing methods of retrieval mechanisms. In Chapter 3, the detailed description of the strategy, how the agents along with matchmaking and brokering protocols are implemented on the grid, in addressing the problem is described. Chapters 4 and 5 present the detailed design of the infrastructure. The implementation details are outlined in Chapter 6, followed by future work and conclusions in Chapter 7.

CHAPTER 2

METHODS, TECHNOLOGIES AND INFRASTRUCTURES FOR SHARING AND EXCHANGE OF MODELS

Today, the industry has moved from a centrally managed environment to a shared collaborated one, so that various organizations can work together and share resources. E-commerce requires services from different organizations or from different resources within a single enterprise over heterogeneous dynamic and distributed organizations for various purposes.

Similar situation exists in the area of military simulation. Because of cost limitations in training individuals, new forms of threat and new techniques for handling warfare [Richard E Hayes 2002], need for military simulation increased vastly. Individual components like tanks, soldiers and airplanes are simulated into different models. The models are integrated to present a virtual environment where users attain knowledge of different operating scenarios and behaviors of various entities operating within the environment.

As domain specific simulation models are constructed and developed, it would be feasible to reuse [Whittman and Harrison 2001] models instead of developing from scratch every time the same model is built. This would significantly reduce effort and improve project efficiency. As such, model repositories [Zeigler 2000] have come into

existence and the models are becoming readily available from several producer sites. But as the number of model repositories increases, discovering and locating available model repositories and models become difficult. The existing methods of locating models lets the users locate models if the user is aware of the existing repositories. Difficulty arises in discovering new model providers. In addition, the users have no means of being acknowledged of the availability of new models.

The models developed are spread across various platforms and across various domains. Maintaining consistency and quality of service during interactions and sharing between such widespread domains is difficult. Standard abstractions and concepts must be defined to address the above problems across distributed systems. Grid technologies seem to be a promising solution to address the above concerns.

Grid computing allows a collection of physical resources from several domains or several entities from a single domain to be encapsulated together into a single virtual computer and at same time achieve desired Quality of Service [IBM Redbooks. 2004]. The virtual computer can be used as any local system by users in obtaining the resources they need. The virtual computer will have rules and policies defined by the provider and consumer of resources regarding shared access to the resources. The grid environment follows the service-oriented architecture.

2.1 Service Oriented Architectures

Service oriented Architecture refers to an architecture, which consists of collection of distributed, heterogeneous components called as *Services* [IBM Redbooks. 2004]. A service is a set of actions encapsulated together from the view of service provider and

service requester [Savas Parastatidis 2003]. The services can be implemented in any programming language, on any platform and operating system.

In a service-oriented architecture the functions and facilities provided by a service are exposed by an interface definition. The users are not concerned with the implementation of the service. Users select a particular service using the operations exposed by the service through its interface. The basic components of service-oriented architecture, shown in Figure 2.1, are as follows

- Service provider: A service provider creates a service, builds the description of the service and advertises the service description in one or more registries and receives and addresses any service request messages from the consumer.
- Consumer: The consumer looks for a service description from one or more registries and invokes the service based on the description provided in the registry.
- Registry: Advertises service descriptions published by the service provider and lets the consumer search for the services contained within it.

An application component can play one or more of the roles mentioned above.

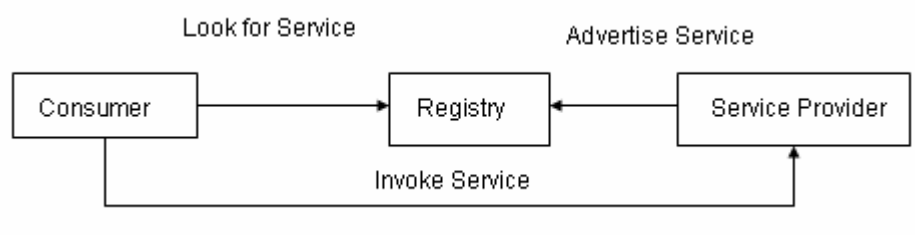

Figure 2.1: Service Oriented Architecture

The grid is a service-oriented architecture and an extension of web services. Web services are the realization of the abstract Service Oriented Architecture (SOA) [IBM Redbooks. 2004]. This forms the basis on which the grid services or the grid environment is built.

2.1.1 Grid Technologies

The term "grid" refers to a distributed computing infrastructure for advanced science and engineering [Foster and Kesselman 1999]. Significant research has been conducted on the construction of the grid infrastructure. In heterogeneous distributed systems, a grid is used for sharing resources and solving problems. A set of individuals or institutions termed as virtual organizations [Foster et al 2001; Foster et al 2002a] control sharing of these resources. The Open Grid Service Architecture (OGSA) provides set of services and protocols that define how to discover the resources, determine who is allowed to access and retrieve the resources at a particular point of time. The protocols define how the resources interact, negotiate and share information. A service is defined by the protocol it speaks and the behavior it implements.

2.1.1.1 What is a Grid?

The grid is built on grid services. Grid services are defined by OGSA, OGSI and GT3.

- OGSA: Open Grid Service Architecture [Sotomayor 2003; Foster et al, 2001] is an open, common, standards-based architecture for grid-based applications. OGSA provides a set of interfaces and each grid service must implement these interfaces. An OGSA defines what a grid service should have.

- OGSI: The Open Grid Service Infrastructure [Sotomayor 2003] defines a formal and technical specification of what a Grid Service is. Grid services are specified by OGSI.
- GT3: Globus Toolkit [Sotomayor 2003] is a software toolkit used for programming grid based applications. OGSI is implemented using GT3.

2.1.1.2 Grid Architecture

The architecture of the grid is organized into several layers [Foster et al 2001]. The architecture encapsulates protocols with common behavior at same layer. Each layer provides a particular service or capability and protocols at higher level depend on the underlying layers. The description of functionality of each layer is as follows.

- Fabric Layer: Fabric layer provides resources that are shared, using the grid protocols. The resources can be storage resources, data resources, network resources etc. A resource can be a logical entity like a distributed file system or a database.
- Connectivity Layer: Connectivity layer provides mechanisms for facilitating communication between resources at the fabric layer. It provides communication and authentication protocols for exchange of data between resources. Communication requirements cover transport, routing and naming. The protocols used at connectivity layer are obtained from the Internet layered protocol architecture, from TCP/IP stack like IP, TCP, UDP, and DNS. Connectivity layer provides authentication like single sign on, delegation, etc.

- Resource layer: Resource layer builds on the connectivity layer to provide secure monitoring, negotiation, control, accounting and payment of sharing operations on individual resources. Resource layer protocols call fabric layer functions to access and control local resources.
- Collective Layer: The resource layer is concerned with issues related to a single resource while the collective layer deals with issues, which are more global like interactions between collections of resources. The collective layer implements different behaviors without additional requirements on the resources being shared. Collective layer protocols are used to coordinate multiple resources.
- Application Layer: The application layer constitutes the user applications that participate in the grid environment. Applications are developed using services defined at lower levels. An Application programmer views the grid as a collection of protocols and services defined at several layers. The application layer defines the application programmer's view [Foster et al 2001].

2.1.2 Open Grid Service Architecture

The Open Grid Service Architecture (OGSA) defines service semantics using concepts and technologies from the grid and web service communities. The architecture also provides standard protocols and mechanisms in creating and discovering the grid service instances, provides location transparency and supports integration with native platform facilities.

The Open Grid Service Architecture is based on two main technologies. The Globus tool kit and the Web services. Globus is a toolkit used for grid-based applications

and web services is the widely used framework for accessing network applications. The grid [IBM Redbooks. 2004] is an extension of web service technologies and inherits several web service properties like service description and discovery, automatic generation of client and server code from service description, binding of service description to network protocols and compatibility with higher level open standards.

2.1.2.1 Globus Toolkit

Globus Toolkit [Foster et al 2002a; Sotomayor 2003] is a community-based toolkit that provides open source set of services and software libraries that are used for creating grid applications. The tool kit deals with all the issues of service discovery, communication, data, resource management, security and portability. Various components of the toolkit related to the grid include Grid Resource Management Protocol (GRAM) [Foster et al 2002], Meta Directory Service (MDS-2) [Foster et al 2002] and the Grid Security Infrastructure (GSI) [Foster et al 2002]. The GRAM protocol is responsible for the creation of transient service instances in a secure and reliable manner. GSI provides all the security measures of authenticity and authorization to remote computations. MDS-2 manages the lifetime of published information using a soft state protocol called Grid Notification protocol [Foster et al 2002].

2.1.2.2 Web Services

Web services [Sotomayor 2003] are a recent technology and uses Internet based standards to achieve distributed computing. It is more efficient than other distributed technologies like CORBA [Ken 1998], Java RMI [Ken 1998] and others. Web services

are independent of programming language, system software and the platform. Web services use three important technologies: - SOAP [W3C 2000], WSDL [Foster et al 2002] and WS-Inspection [Foster et al 2002]; SOAP is used for communication and exchange of messages, WSDL is an XML Document used for describing web services and WS-Inspection an XML document which provides a collection of service descriptions provided by the service provider.

Grid Services address issues that arise while discovering and locating models. In addition they also address issues when new models become available.

2.2 Agents

As consumers and service providers contact each other and share information about the models over the grid, software programs called "Agents" [Sycara et al 2002a] handle the interactions between them. Agents facilitate partial matching, provide location transparency and recommendations. In Agent Based Software Engineering [Genesereth 1994], software programs on several heterogeneous, distributed systems are represented as agents [Retsina 2001; Sycara et al 2002a] and they interact among themselves through exchange of messages using agent communication language [Sycara et al 2002; Finn et al 1993; Durfee et al 1997]. Agents exchange complex data and logical information using agent communication language [Sycara et al 2002].

The term "agent" is defined in different ways [Franklin et al 1997; Woolridge et al 1995; Weiss 1998; Franklin et al 1997]. An agent is software or a hardware entity that performs some specific operation for its users. It is autonomous, reactive and proactive and interacts with users and other agents towards achieving its goals. A goal defines the

activities an agent has to perform. Each agent is assigned a role, which defines how the agent interacts with other agents to achieve the goal. A context defines an agent's role and goal. An agent's role changes when it is in a different context [Sycara et al 2002a].

An agent may possess any of the following characteristics: - autonomy, commitment [Genesereth 1994], Cooperation [Foner 1993], reactive, proactive, social ability, intelligence [Weiss 1998], goal oriented, learning, and communicative [Franklin et al 1997]. Agents are being widely used in many domains like book buying auctions, mobile communications, workflow management, network management, air traffic control, data mining, electronic commerce [Hyacinth 1996], data collection and filtering, pattern recognition [Croft et al 1997] and database integration and in various institutes for software collaboration [Genesereth 1994]. Many institutes, organizations and corporations are investing significant effort in agent's research [Retsina 2001; Durfee et al 1997]. The agents are classified based on their characteristics or the roles they play. Table 2-1 depicts a high-level classification.

The concept of information agents, middle agents and interface/ user agents are applicable in this work. Using middle agents can facilitate partial matching of models and realize the notification/subscription facilities. A Hybrid agent (combination of two or more agents) of interface agent and information agent can be used to address how the consumer will interact with the service providers.

Table 2-1: Types of Agents

Agent	Description
Reactive agents	Simple agents that act based on stimulus/response to the current state of the environment. [Weiss 1998]
Mobile agents	Mobile agents move from one system to the other in the network. They improve performance by moving the agent to the data than moving data to the agent. [Croft et al 1997]
Collaborative agents	Autonomous agents that interact with other agents and share information for various purposes. [Hyacinth 1996]
Interface/User agents	These agents interact with the users and provide assistance in using a particular application. [Hyacinth 1996]
Middle agents	Agents that help locate other agents are middle agents. [Sycara et al 2002] The other agents submit requests to these agents for some information and these agents retrieve information about other agents that can satisfy the request.
Information agents	Information agents are responsible for managing information obtained from distributed resources. [Foner 1993]

2.2.1 Matchmaking Agents

Matchmaking [Decker et al 1996] facilitates the exchange of dynamic and diverse information between information providers and requesters in an effective and efficient way. If the user is aware of the existence of the provider she/he can contact him without further assistance. But this is difficult in open dynamic environments since accurate knowledge of the location of service providers is extremely difficult to gather, as providers can leave or enter the environment frequently.

The alternative approach is the use of matchmaking or middle agents or matchmaker [Zhang Z, Zhang C 2002]. Matchmaking is a cooperative partnership between information providers and consumers assisted by an intelligent facilitator

[Kuokka and Harada 1995] called the matchmaker. A matchmaker is responsible for finding an appropriate information provider suitable for a given request. An information provider advertises his capabilities to the matchmaker. The matchmaker stores these advertisements. The requesters make requests to the matchmaker for the desired information and the matchmaker finds appropriate providers that satisfy the requests using different efficient algorithms. Matchmaking is essential in open dynamic environments, where providers or requesters can enter and leave the environment as they desire.

The main advantage of Matchmaking is that consumers and providers can retract or issue requests and advertisements in a dynamic way whenever they desire. This way the information doesn't become stale [Kuokka and Harada, 1995]. Another advantage of matchmaking is that it provides the users with the ability of playing an active role in information retrieval. The performance of the matchmaking process relies on the efficiency of the algorithms employed for matchmaking.

Significant research has been performed with regard to middle agents as well as the matchmaking process. Several matchmaking algorithms have been developed. Every technique emphasizes on a different criteria to perform matchmaking. One of the earliest matchmakers is the ABSI facilitator [Singh, 1993]. Two matchmakers SHADE System and COINS System were presented in [Kuokka and Harada 1995]. These matchmakers are agents and communication is achieved using agent communication language. [Zhang Z, Zhang C 2002] argues, the performance of the information provider in satisfying the request must be considered when matching the advertisements with requests. The

technique employed here performs matchmaking by considering the performance of the information provider in solving problems in the application domain.

Most of the existing matchmaking mechanisms are key word based in that they retrieve information providers whose description exactly matches the requester's description of the information. However, problems arise if the vocabulary used by the requester is different from one used by the provider. A service provided by the information provider could be represented in one way and the same service requested by requester may be represented in a different way. In such cases, they are not matched although both are semantically the same. A mechanism that can perform syntactic as well as semantic matching is desired.

Ludwig and Santen present the inexact or partial matching process. The matchmaking mechanism retrieves providers that provide services that partially match [Ludwig and Santen 2002]. The matching engine provides flexible matches, ones that have some degree of similarity between advertisements and requests. Partial matching is also been applied in [Kawamura et al 2002; Sycara et al 2002]. The algorithm used here in [Kawamura et al 2002] compares inputs and outputs of the request with each advertisement and determines the matched set. The matched set is ranked based on the degree of match and resultant set is returned to the requester. The matchmaking process provides exact match, plug-in match (less accurate but useful match) and the relaxed match (the least accurate match). Partial matching has also been performed in [Sycara et al 2002]. The matchmaking is performed in 5 filter stages: - Context, profile, similarity, signature and constraint.

The matching should not be based on just keyword retrieval. It should be automated, accurate, efficient and effective [Sycara et al 2002]. In addition it would be feasible if a common vocabulary or language were made available. The service providers and requesters can describe their capabilities and requests using the common vocabulary. The use of common vocabulary would resolve the semantic matching issues. The above concerns are addressed in this work.

2.2.2 Brokering Agents

The broker agent or the brokering protocols are responsible for contacting the service provider, communicate the requests to the provider and deliver the results back to the requester. In other words a matchmaker retrieves the list of service providers that satisfy a particular request, while the broker agent implements a brokering protocol to contact the information provider. A protocol is defined as a pattern of message exchanges [Durfee et al 1997].

The broker provides transparency between requesters and service providers by accepting requests from the client, transferring the request to a capable service provider and returning results to the requester. The broker provides communication and location transparency for distributed applications [Hayden et al 1999].

The concept of broker agent or brokering protocols is an ongoing research [Decker et al 1996; Fensel 1997]. Brokering has been used in ABELS [Murphy et al 2003]. Various brokering protocols are defined and used as part of message exchanges

between the agents within its architecture [Durfee et al 1997]. Protocols define how two agents or software components interact or talk with each other. They are independent of the algorithms implemented by those agents, are unbiased and stateless.

[Kuokka and Harada 1995] uses a matchmaker agent to perform matchmaking. The agent uses various knowledge sharing brokering protocols to interact and provide information to users. The brokering protocols used here are recommendation, recruiting, brokering and content based routing.

The content based routing or publish/subscribe [Muhl et al 2002] mechanism provides users up to date information as soon as it is published. This type of protocol is useful in e- commerce and e-business where information changes dynamically and up-to-date resource and data information is of utmost importance.

In this work the concepts of grid and agent technology along with matchmaking and brokering mechanisms are combined together to develop the architecture that addresses all the issues mentioned above.

Chapter 3

Active Model Retrieval on the Semantic Grid

This chapter presents a detailed explanation of the strategy followed in building the active model retrieval mechanism over the grid infrastructure. A brief description and functions of each component of the application is presented here. Next, the interactions that take place between the clients (service providers and consumers) and the system are described. The various types of brokering protocols are described next, followed by the algorithms used to perform matchmaking. The application is built as a client/server application and significant part of the proposed infrastructure was built on the server. The entire application has its basis on the grid infrastructure. The server and client applications are deployed on the grid. The main modules of the architecture on the server are presented in the Figure 3.1.

The components on the server constitute the communication module, matchmaking engine, ontologies, advertisement database, and the registry. The communication module forms the entry point for the server. All the requests to the server pass through the communication module. The communication module is responsible for interacting with all modules on the server and generating responses based on the requests.

3.1 Knowledge Base

The ontologies [Gruber 1993] form the knowledge base in the architecture. A service provider, who provides the military models, advertises a model with a description in his own vocabulary. A consumer, who wishes to access the models, makes a request for the same model with a description of his own. Although both represent the same model, the matchmaking process may not match the request and the advertisement. This discrepancy is addressed by the use of ontologies. Ontology provides a common vocabulary, by which both service providers and consumers can share information.

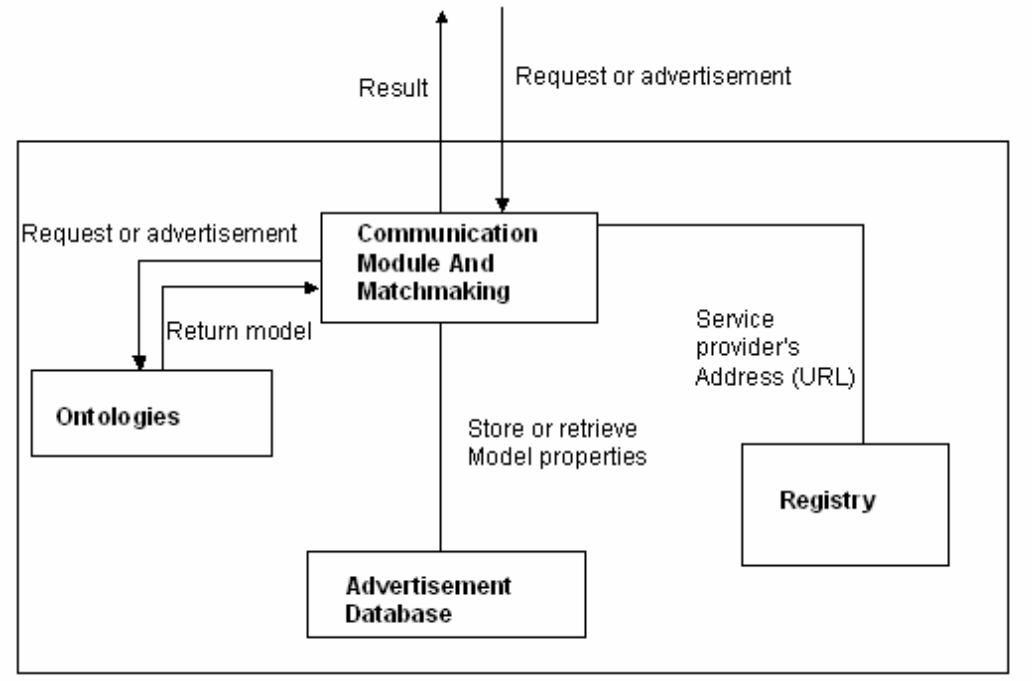

Figure 3.1: Significant modules of the architecture

The ontology classifies the domain knowledge into various levels. As such, the ontology facilitates knowledge sharing and reuse [Gruber 1993]. It classifies and defines

each of the models and presents them in a hierarchy along with their attributes, to provide a common foundation, on which the service providers can describe models, and the consumers can select the models. Each advertised model conforms to the definitions of that model in the ontology. Based on the definitions and classification given in the ontology, the consumer selects the model.

3.2 Service Provider Agent

The service provider selects the ontology and the desired model from the ontology hierarchy. Next, the model along with its information is published into the advertisement database, which is deployed, on the server. The service provider interacts with an interface agent, which acts on his behalf and works with any requests and responses from the server. The personal information of the service provider, like name and contact information along with the model's information and attributes, is placed into the advertisement database. The information on how to access the models is placed into a registry. This provides considerable abstraction between the information about models, and the information that helps access the model.

3.3 Consumer Agent

The consumer, that requires the model, interacts with the interface agent to retrieve information about the model it needs. The data flow between the consumer and the application is shown in Figure 3.2. The consumer interacts with the user/interface agent in two modes. Push Driven mode and Pull Driven mode. In push driven mode the consumer makes a request for a model, and the matchmaking and the brokering schemes

perform searching and retrieval of models. The request for subscription for future models, and subsequent notifications by the server, forms the pull driven mode.

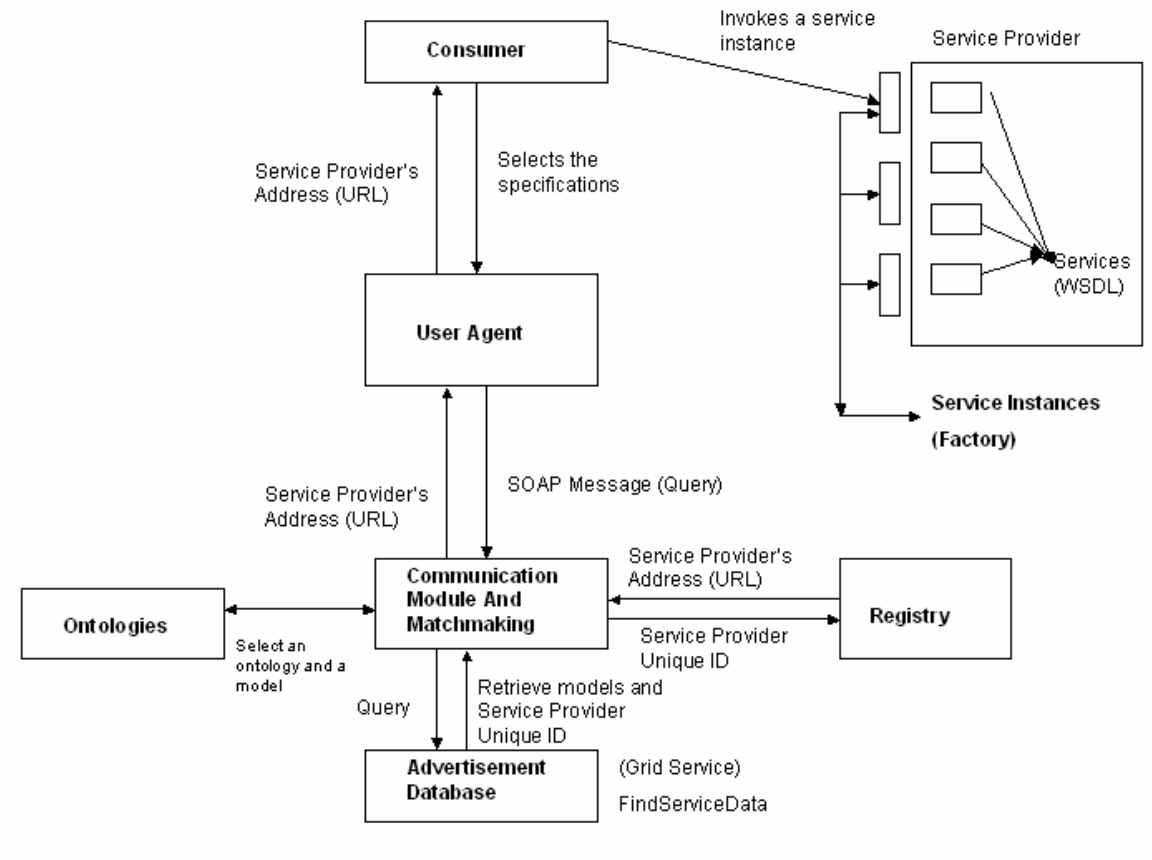

Figure 3.2: Data Flow Mechanism

3.4 Brokering Protocols

The consumer agent is responsible for interacting with the server, retrieving the desired models, and then contacting the brokering agent. The brokering agent is responsible for implementing the brokering protocols. The consumer agent presents the consumer with the list of ontologies and various brokering protocols for selection. The consumer agent

passes the selected model to the communication module on the server, and delivers the matched models back to the consumer. The response is presented to the consumer in different ways based on the brokering protocol selected by the consumer. The various brokering protocols that will be implemented here are as follows:

- Recruiting: The consumer agent asks the matchmaking engine to select a suitable service provider, and send its request to that provider. However any subsequent responses from the service provider are sent directly to the consumer agent. In this protocol, the matchmaker agent selects from the matches found, the most suitable service or model, and redirects the request to that service provider. Recruiting provides the service provider agent with the consumer's Address (URL), so that it sends the results directly to the consumer agent. The interactions are shown in Figure 3.3.
- Recommendation: The consumer agent asks the matchmaking engine, to perform matchmaking and retrieve the list of matched models. The consumer agent presents the list of models to the consumer, who selects a desired service provider. The consumer agent then invokes the brokering agent, to contact the service provider agent. The interactions are shown in Figure 3.4.
- Notification/Subscription: The brokering agent subscribes to a particular model and is notified, when that particular model is available or published by a service provider. The interactions are shown in Figure 3.5.

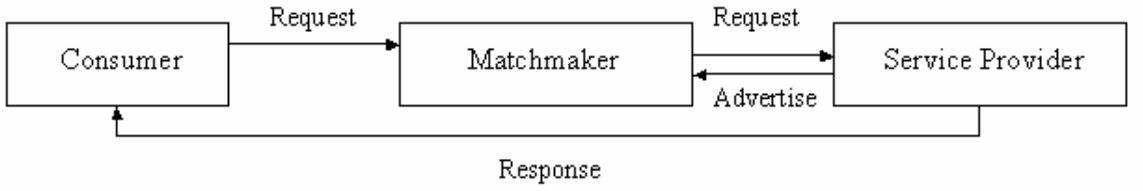

Figure 3.3. Recruiting

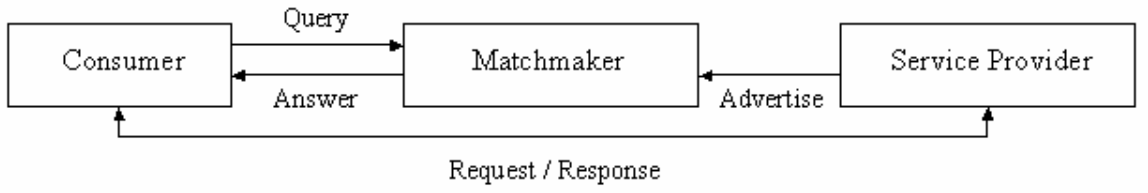

Figure 3.4. Recommendation

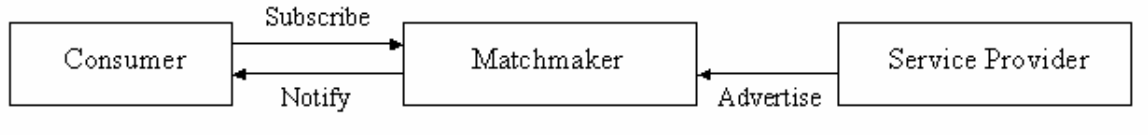

Figure 3.5. Notification/Subscription

3.5 Matchmaking

On the server side, the matchmaking engine performs the matching between the consumer requests and the service provider's advertisements. The matchmaking engine performs the filtering of models in two stages: - Concept level Matching and Tree Pattern Matching.

3.5.1 Concept Level Matching

In Concept level matching, the desired model is searched in the advertisement database. In addition to the desired model, all the models that are at upper levels above the desired model, in the ontology, are searched for. All the models that are at lower

levels below the desired model, in the ontology, are also searched for. These constitute the partially matched models or "sufficiently similar" [Paolucci et al, 2002] models.

Figure 3.6 shows a section of the simulated models ontology. It pictorially describes how the models are classified and presented as a hierarchy. When a "Tank" model is searched, a search for generalized as well as specialized models is made. The generalized and the specialized models constitute the partial matches.

The algorithm for first stage of filtering is as follows:

```
Match (request)
        {
                Matchmodels= null
                For each adv in advertisements do
                {
                If match (request, adv) then
                Matchmodels.add(adv)
                degreeOfMatch(request,adv,ontology)
                }
                return sort(Matchedmodels);
        }
```

The models are ranked, based on the distance between the desired model and the advertised model, in the ontology. The degree of match is computed as follows. The algorithm returns Exact, Substitute, or Partial degree of match.

```
degreeOfMatch(request,adv,ontology)
        {
                if request=adv then return Exact
                if adv subclassOf request then return Substitute
                if adv superclassOf request then return Partial.
                otherwise null
        }
```

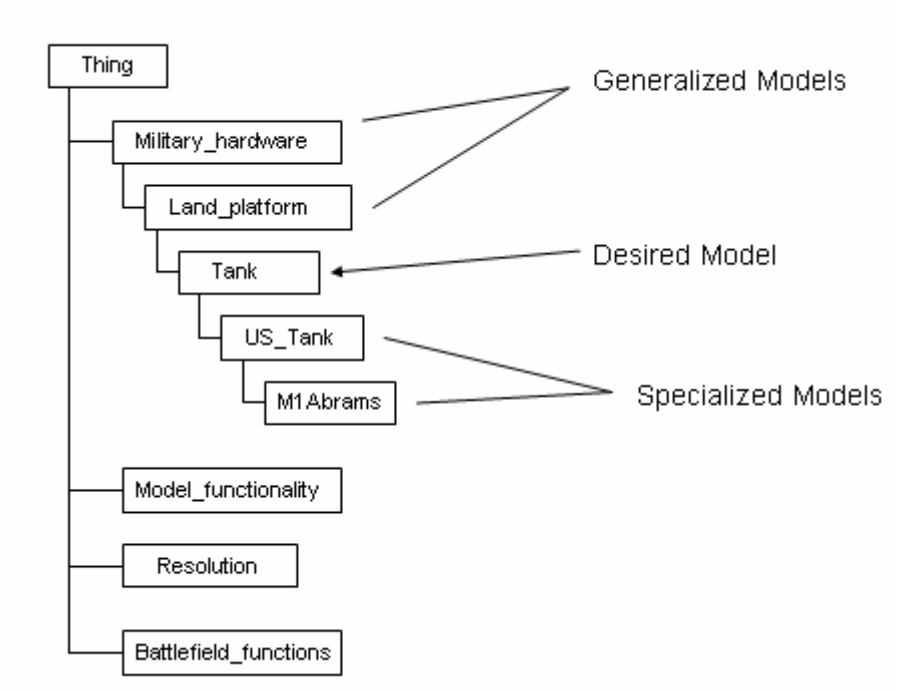

Figure 3.6. Section of the Military Models Ontology

3.5.2 Tree Pattern Matching

The second stage of filtering forms the tree pattern matching. Once the list of service providers is retrieved, the second filtering stage is performed. The tree structure of the requested model, and the tree structure of each of the models in the list are compared, and the cost is obtained. The degree to which the trees match is computed. The degree of match determines the extent to which the requested model is structurally and linearly similar, to the service provider model. The second phase of filtering is divided into two levels: -linear distance metric and structure metric. The linear distance metric calculates how many attributes match, and the structure metric measures similarity in structure or hierarchy.

3.5.2.1 Linear Distance Metric

The algorithm presented in [Yilmaz 2002] is employed to perform the first stage of tree pattern matching process. The degree of similarity between the models is computed linearly, by transforming the model trees into arrays. A preorder traversal is performed on the consumer and the service provider trees, to transform them into linear arrays of their attributes. The cost of transforming service provider model array into consumer model array is determined, which is equal to the degree to which the two models match. In the algorithm, each operation insertion, deletion, and move are assigned different costs, and the number of deletions, insertions, or substitutions needed to be performed, to transform the service provider model array into requested model array, is obtained. If $Cost_i$, $Cost_m$, $Cost_d$ are costs for insertions, deletions, and moves respectively and N_i ,N_d ,N_m are the number of insertions, deletions, and moves respectively, and Lstream is the length of the larger model array, then the linear metric is calculated as:

$$\text{Linear Metric} = \frac{Cost_i * N_i + Cost_m * N_m + Cost_d * N_d}{\text{Max}(Cost_i, Cost_d, Cost_m) * L_{stream}}.$$

The algorithm is as follows [Yilmaz 2002]:

Assume C is a constant cost for any move or substitution operation. Let the attributes in consumer tree be a_0, a1…..a_m and attributes in service provider tree be b_1, b2….b_n.

```
d (a0, b0) = 0;
for i=1 to m do d(ai,b0) = cost(delete(ai))
for j=1 to n do d(a0,bj) = cost(insert(bj))
for i=1 to m
        for j=1 to n do
                If ai = bj then cost (move (ai, bj)) = C
                else
                cost(move (ai, bj)) = cost(delete(ai)) + cost(insert(bj))
                d(ai, bj) = min { d(ai -1, bj -1) + cost(delete(ai)),d(ai -1,bj -
                1) + cost(move(ai, bj)), d(ai, bj -1)+cost(insert(bj)) }
return d(am,bn)
end
```

Figure 3.7 shows how linear distance metric is calculated for the service provider and the consumer model arrays. The trees are traversed and transformed into linear arrays and presented in the figure. The transformation of service provider model into consumer model is shown. “Ammunition_storage”, “loader_weapon”, and “power” match in the two models. The attributes “Grenades”, “Rounds11400”, and “Rounds120mm” are inserted into the service provider model array. The attributes “Coaxial_Weapon”, “barrel”, “rounds”, “acceleration”, and “speed” are deleted from the array. The insertions and deletions to be performed, to transform the service provider model into the consumer model, therefore are N_i=3 and N_d=5 respectively.

The cost of insertion, $Cost_i$=2 and the cost of deletion, $Cost_d$=1. The high insertion cost indicates that the attributes of the qualified or the requested model are missing. The extra attributes are simply deleted and are not penalized as missing attributes. The length of the larger model array is L_{stream}= 8, then the linear distance metric is calculated as

$$\text{Linear Distance Metric} = \frac{Cost_i * N_i + Cost_d * N_d}{\text{Max}(Cost_i, Cost_d) * L_{stream}} = 3 \frac{* 2 + 1 * 5}{2 * 8} = 0.69$$

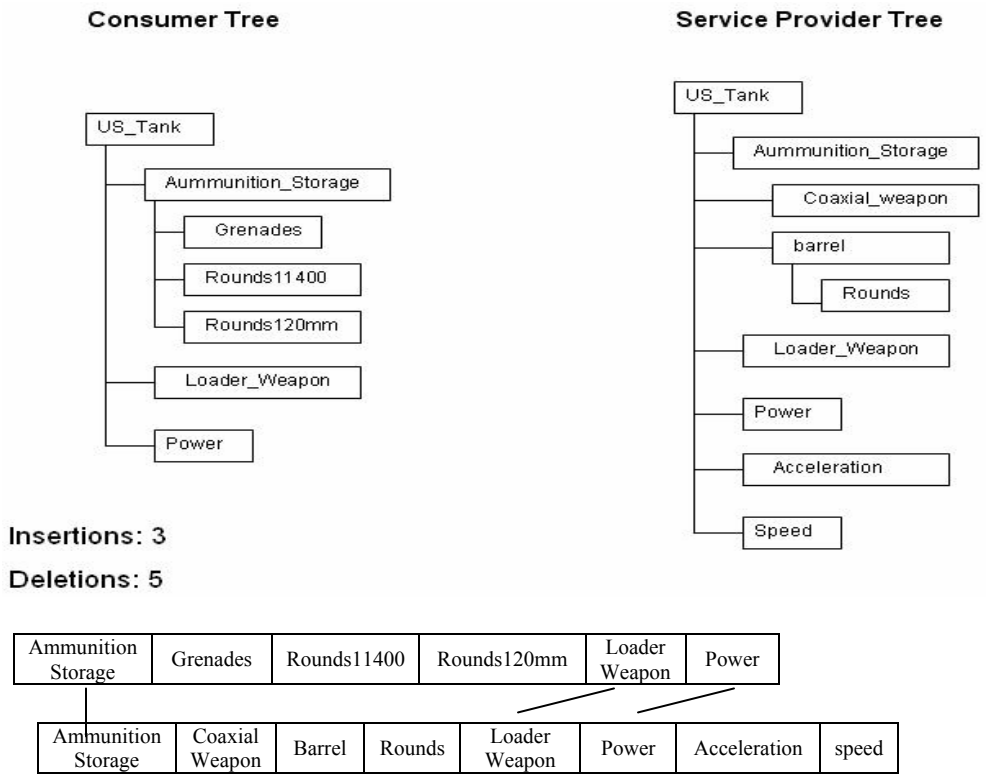

Figure 3.7 Calculation of Linear distance metric

The linear distance metric is calculated for every pair of consumer, service provider models. The pair of models, that have the least linear distance metric, have strong correspondence and similarity than the other models.

3.5.2.2 Structure Metric

The structure metric determines the degree to which the two trees are structurally similar. For each pair of the trees, a structure metric is calculated to determine the degree of similarity between the structures. The algorithm used is based on the one presented in [Romanowski and Nagi 2005]. In the approach, the two trees are divided into groups of single level subtrees and terminal subtrees. A single level subtree is a parent node with its

children. The children may or may not be leaf nodes. A terminal tree is a parent node with its children, which are all leaf nodes. Assume there are two trees T_a and T_b as shown in Figure 3.8.

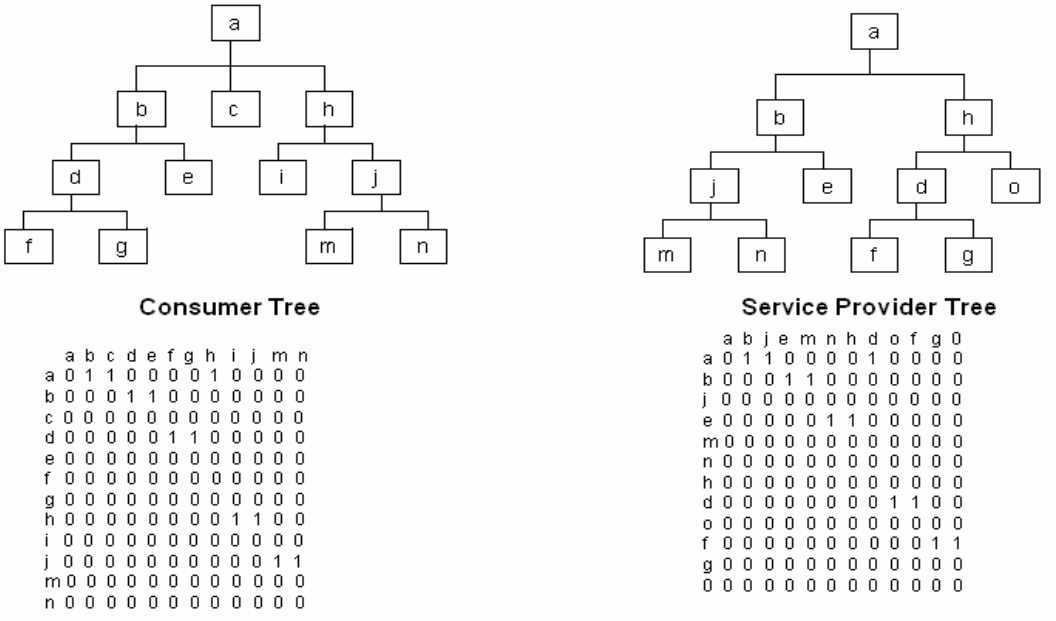

Figure 3.8 Trees and their corresponding adjacency matrices

An adjacency matrix is calculated for each of the trees. An adjacency matrix is a two dimensional matrix where the root and the nodes form the rows and columns. If the node in the column is a child of the node in the row, their corresponding entry will have a "1" or else "0". If the sizes of the two matrices differ, the smaller matrix is filled with rows and columns of 0's, so that the two matrices are of the same size. After the adjacency matrices of T_a and T_b are calculated, as shown in Figure 3.8, each tree is grouped into pairs of terminal subtrees and single subtrees. Next, each subtree in T_a is compared with every subtree in T_b. In trees that exactly match, the children nodes are removed from the subtrees. As shown in Figure 3.8, the trees at d, j matches. The

children f, g, m and n are removed from the trees. The trees are shown in Figure 3.9 after removing the exact matches.

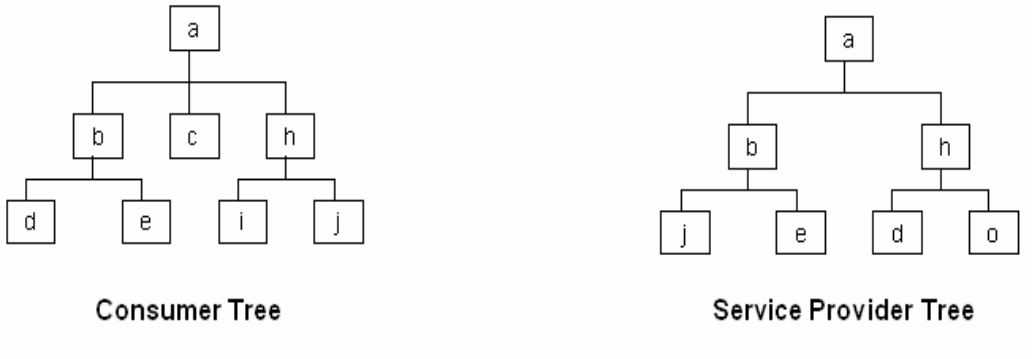

Figure 3.9. Reduced consumer and service provider trees

From the remaining subtrees, the distance between subtrees of T_a and each subtree of T_b is calculated. For each subtree of T_a, the subtree of T_b that has the minimum distance is obtained. The distance between trees rooted at b in both trees is 2, as "e" matches but d and j do not match. Similarly, the distance between tree at b in $T_{a,}$ and tree at h in $T_{b,}$ is also 2. Hence the minimum distance for tree rooted at b in T_a is 2. The minimum distance for tree rooted at h is 2. The distance for the tree rooted at a, is 1 as b and h match but c does not. The structure metric for T_a and T_b is the sum of distances between all these pairs, divided by n^2, where n is the number of nodes in the larger tree. In the example, the distance is $(2+2+1)/ n^2$, where n is 12, so structure metric = (2+2+1)/ 144 = 0.035. The structure metric is calculated for each consumer and service provider pair, and the one with least value forms the best match. The algorithm is presented as follows:

Input: Unordered trees T_a, T_b, $| T_a | <= | T_b |$
Output: Structure Metric determining the distance between two trees T_a and T_b.

1. Form Adjacency matrices of T_a and T_b, A and B using preorder traversal.
2. If |B| > |A| then augment A with |B|-|A| rows and columns of zeros at the right end and bottom of the matrix.
3. calculate the initial structure metric Sab = D / n^2 where D=|A-B| where D represents all nodes that are in A and not in B and nodes that are in B but not in A. n is the number of nodes in the larger tree.
4. Find exactly matching subtrees.
   For i = 1 to m where m is the number of subtrees in B
   Start at right side of matrix B and move towards left to the root.
   Find the first terminal subtree tb_i rooted at n
      Search A for an exactly matching terminal subtree t_a with root n
      If ta and tb_i are exactly matching subtrees then reduce both A and B by removing the rows and columns corresponding to the child nodes of ta and tb_i.
   Next i.
   Repeat for single level subtrees in A and B. If no more exact matches remain, Continue.
5. find paired subtrees using TreeMatch(T_a,T_b). Get D from TreeMatch.
6. calculate $S_{ab} = D / n^2$.
7. Return S_{ab}.

Function TreeMatch (T_a, Tb)

1. Get all subtrees of T_a and T_b with a minimum distance between them.
2. D = Sum of min(t_a,t_b) for all subtree pairs {t_a}, {t_b}
3. return D.

Once the consumer returns with a list of service providers and the selection of service provider is made, the next step is to query the repository at the service provider site. The consumer enters the values of the model attributes he is interested in. Next, the repository at the service provider site is queried for that model, which satisfies these attributes, and the model along with the attribute values are returned back to the user.

Chapter 4

The Design of the Agents Used in Active Model Retrieval

This chapter presents the architectural and detailed designs of the Agent-Mediated Brokering and Matchmaking System (ABMS). The functions of the ABMS are presented in terms of structural and behavioral models. The static view of the application is presented using the class diagrams, and the dynamic view is presented using the use case diagrams, sequence diagrams, and the state chart diagrams. Chapter 4 presents the deployment diagram, class diagrams, state chart diagrams and use case diagrams followed by Chapter 5, which presents the sequence diagrams.

Figure 4.1 shows the deployment diagram for ABMS. The three dimensional boxes called nodes, represent three different machines. The grid server node is where the matchmaking engine and the database server components reside. The grid consumer client contains the user agent and the brokering agent components. The grid service provider client contains the user agent. Each of these client nodes interacts with the matchmaking engine component that resides on the grid server node. The matchmaking engine interacts with the database server, as well as the ontologies and address requests from the two clients.

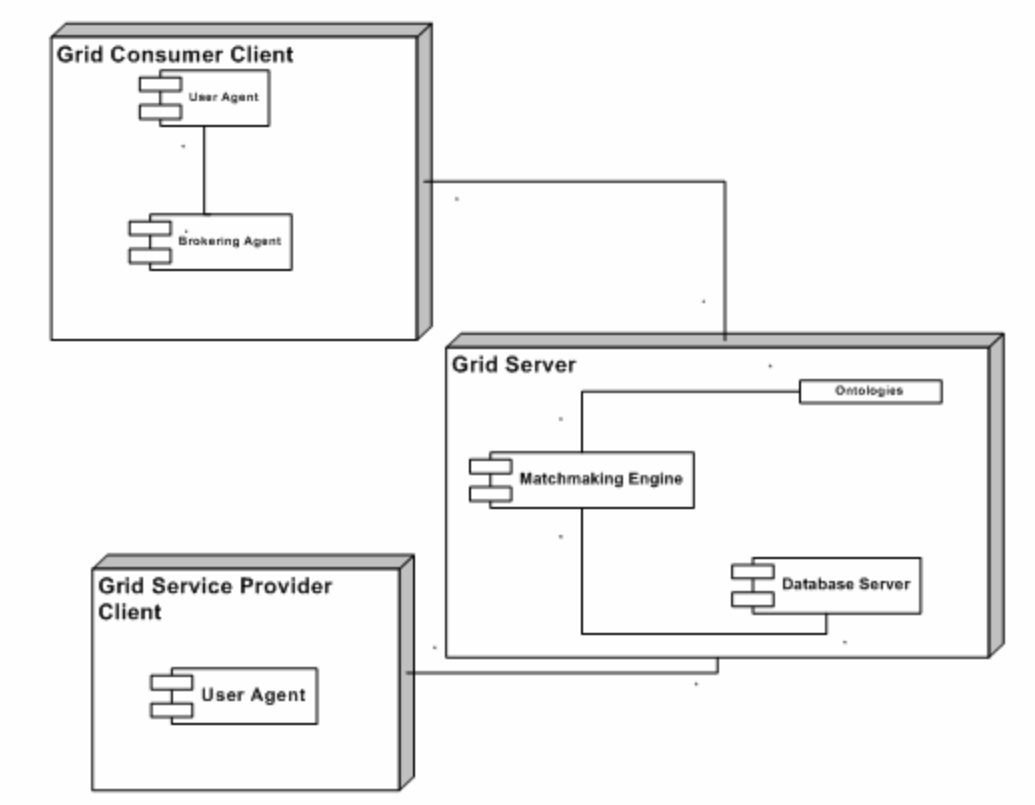

Figure 4.1: Deployment diagram

The use case diagram presented in Figure 4.2 presents the user perspective of ABMS. The various functionalities provided by the system and the interactions between the users and the system are presented in the diagram.

The various classes used in the application, as well as the relationships between them, are presented using the class diagrams. The class diagram for the entire application is shown in Figure 1 in the appendix. The significant classes, by which important application logic is performed, are OntologyTreeGUI, Comparingtrees, Structure, GridServiceDQLQuery, Matchmaker and CallNotifyGridService.

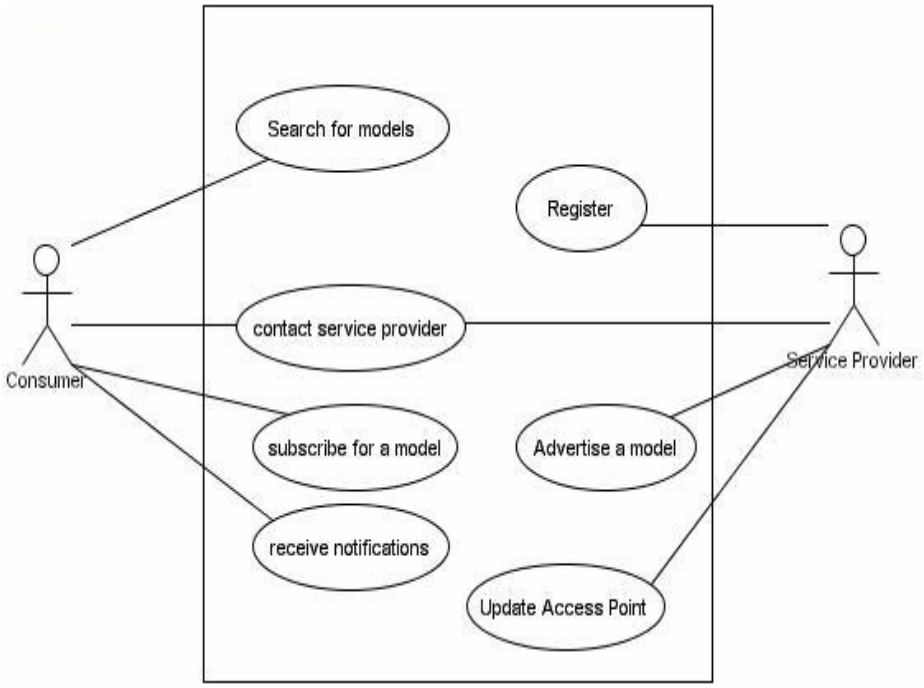

Figure 4.2. Use Case Diagram

The application has four subsystems: the consumer agent, the service provider agent, the matchmaking agent and the brokering agent. The functionalities of each one of the subsystems are described as follows.

4.1 Consumer Agent

The tasks performed by a consumer agent are described as follows:

- The consumer agent takes the input from the consumer, the selected ontology and the selected protocol.
- It parses the selected ontology and presents the hierarchical structure of ontology to the user.
- If the selected brokering protocol is recommendation or recruiting, it takes the user's input and queries the matchmaking service on the server.
- Then the consumer agent displays the matchmaking results to the user.

- If the selected protocol is notification, the agent contacts the brokering agent to call the subscription grid service.
- Finally, the agent displays the available notifications, to the consumer.

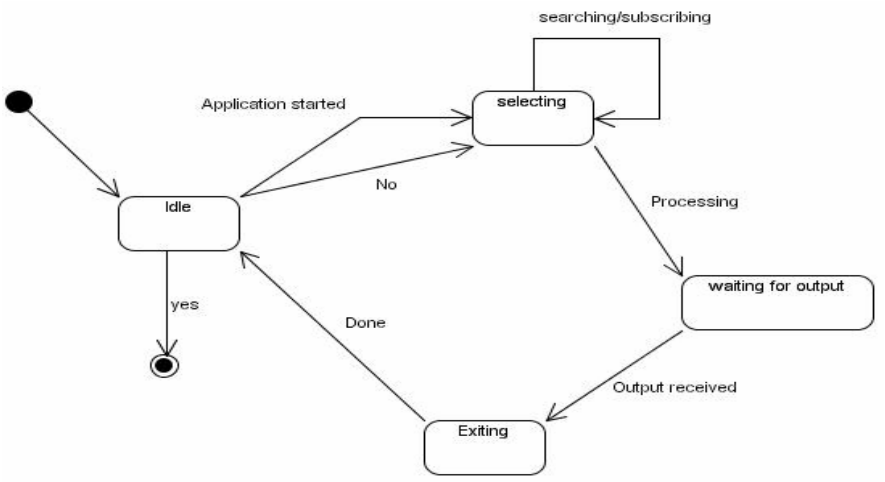

Figure 4.3. Consumer Agent State Chart Diagram.

The various states, a consumer agent can be in are illustrated using a state diagram, in Figure 4.3. In the "selecting" state, the consumer agent takes input from the consumer. The consumer selects either a search or a subscription, for a model. Then the agent moves to the "waiting for output" state where it waits for the results from the matchmaker agent. Once the matched results are obtained from the matchmaker, the consumer agent displays them to the consumer, and moves onto the "exiting" state.

The consumer agent is designed using OntologyProtocolGUI, OntologyTreeGUI, SPProviderListGUI, propertiescheckbox, NotificationScreen and modeltree classes. The relationships between some of the significant classes are shown in the following class diagram. As shown in Figure 4.4, the "OntologyTreeGUI" takes the consumer's choice of ontology and loads the ontology. Then, it obtains the selected model input from the consumer and queries the matchmaker for matched models. It then, presents the list of

service providers to consumer using “SPProviderListGUI”. This class diagram also shows the interactions that take place when the consumer selects recruiting protocol. “SPProviderListGUI” calls the brokering agent’s “GridServiceDQLQuery” class to contact the service provider agent. The relationships between remaining classes are depicted in Figure 5 in the appendix.

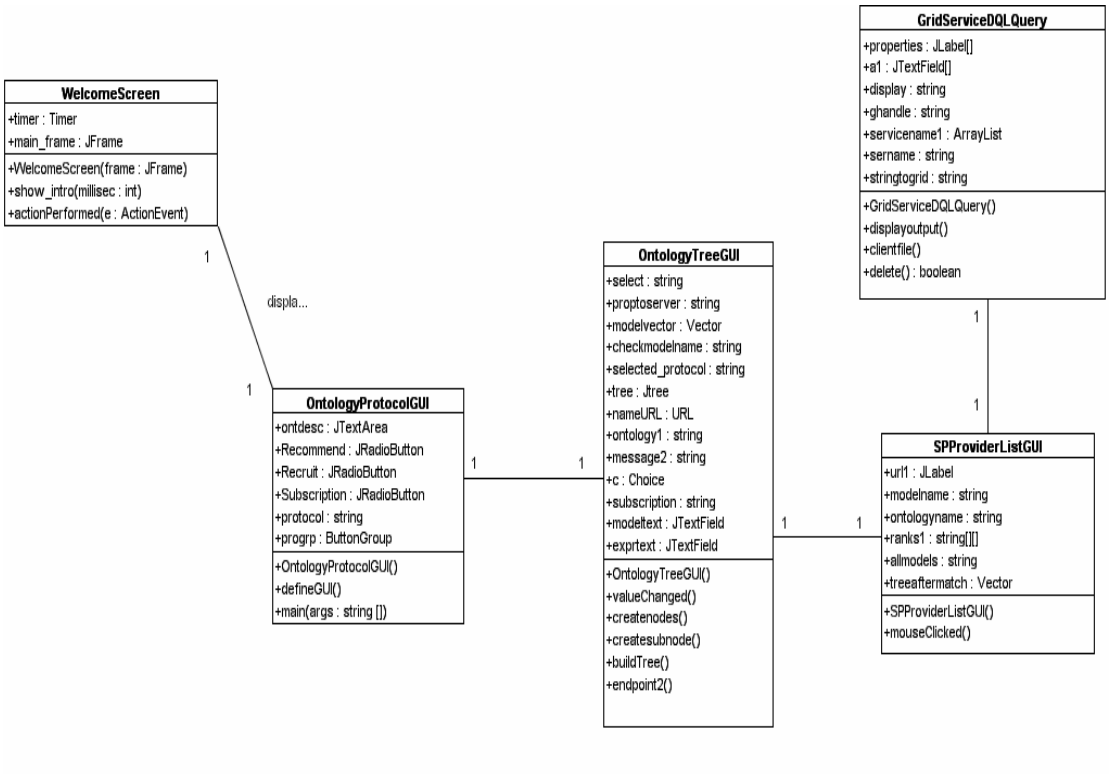

Figure 4.4 Consumer Agent Class Diagram

4.2 Brokering Agent

The brokering agent is responsible for performing the following functions:

- The brokering agent takes the selected protocol value from the consumer agent.
- If the selected protocol is recommendation, then the brokering agent contacts the grid service of the service provider, selected by the consumer.

    - The brokering agent takes the values of the model attributes and forms a DQL query.
    - Then it calls the service provider's grid service client with the DQL query as its parameter.
    - The client calls the service provider's server grid service, retrieves the model instances and sends the response back to the brokering agent. The brokering agent then sends the model instances back to the consumer agent.
- If the selected protocol is recruiting, the brokering agent calls the matchmaking agent on the server, to contact the service provider agent. The matchmaking agent performs the rest of the process of obtaining the model instances.
- If the selected protocol is subscription, the brokering agent subscribes to the selected model on consumer's behalf and calls the notification client grid service to start listening to subscriptions.
- It contacts the consumer agent when a notification is available.

The various states, a brokering agent can be in are illustrated using Figure 4.5. When the consumer selects the protocol, the brokering agent enters into the "Processing" state. Based on the protocol selected, the brokering agent moves to different states. When the protocol is "recruiting", the brokering agent calls the matchmaker for further processing and enters into the "exiting" state. When the protocol is "recommendation", the agent calls the service provider grid service, and waits for model instances from it. When the protocol is "subscription", the brokering agent calls the notification grid service client to subscribe for the model, and enters into "listening for notification" state. Once it receives a notification, it enters into "stop listening" state.

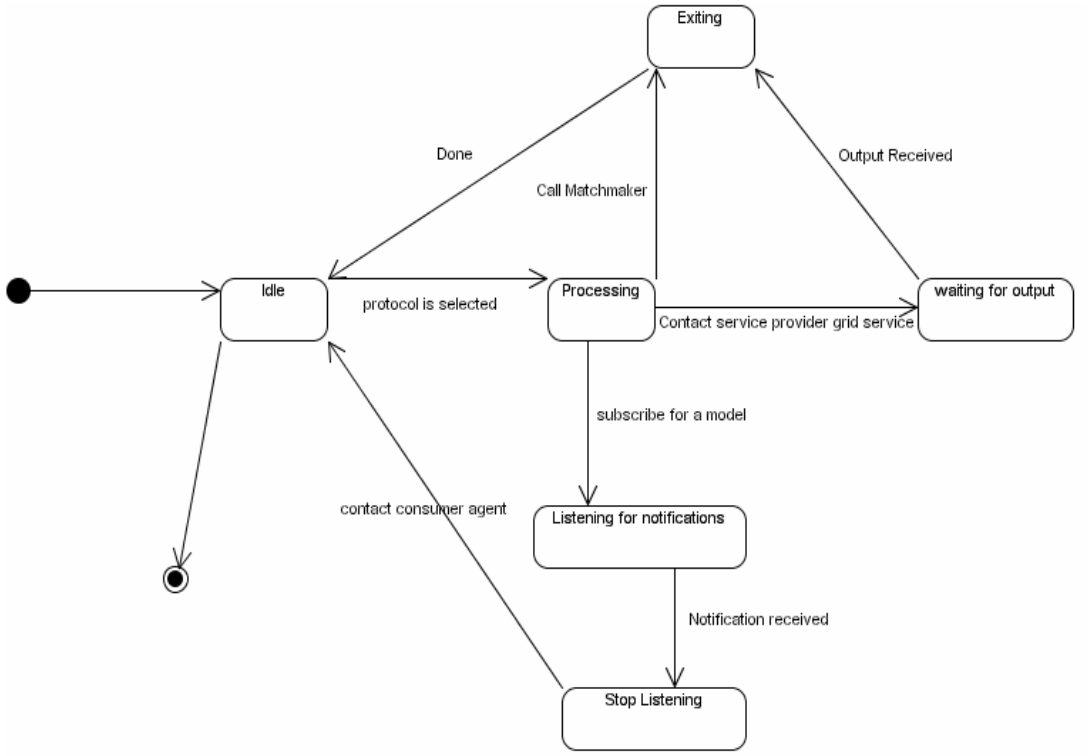

Figure 4.5 Brokering Agent State Chart Diagram

The classes that perform the functionalities of the brokering agent are GridServiceDQLQuery, SubscribeAndListenNotification and NotifyClient. The "GridServiceDQLQuery" class will contain the application logic for the recommendation and recruiting protocols and the "SubscribeAndListenNotification" and the "NotifyClient" will deal with the computation required for notification protocol. The relationships between these classes are shown in Figure 3 in the appendix.

4.3 Matchmaking Agent

The tasks performed by a matchmaker are described as follows:

- The matchmaking agent performs concept level matchmaking, contacts the database server, and retrieves the models that match the desired one.

- It then performs the tree level matchmaking and computes the linear distance metric, and structure metric for each model, matched from concept level matchmaking.
- It also calls the client associated with service provider's grid service, to retrieve the model instances, and returns the model instances back to the brokering agent.
- It receives the model advertisements from the service provider agent and stores the advertisements on the database server.
- It also calls the notification grid service to notify the subscribing client when subscribed model is available.

The various states, a matchmaker agent can be in are illustrated using a state diagram, in Figure 4.6. The matchmaker agent moves into "Waiting for input" state when the server starts. When the matchmaking agent receives input from service provider agent, it moves into "Storing into database" state. It checks for available subscriptions and enters into "Notifying Consumer" state, where it notifies subscribed consumers, of the availability of model.

The matchmaker agent moves from "Waiting for Input" state to "Perform matching" state, when it receives input from the consumer agent. It then contacts the service provider agent, if the protocol selected is "recruiting". It obtains the model instances from the service provider agent. Then it returns the model instances results back to the brokering agent, which returns the results to the consumer.

The classes that constitute the matchmaking agent subsystem are Matchmaker, structure, comparingtrees, IsModelSubscribed and CallNotifyGridService. The relationships between some of these classes are shown in the following class diagram.

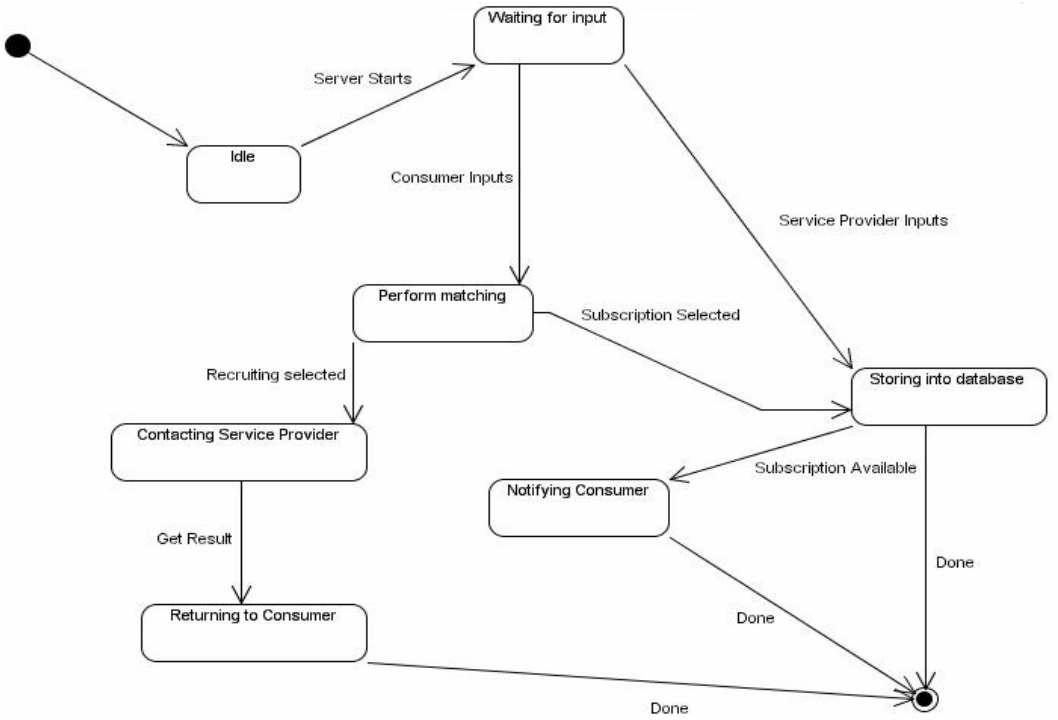

Figure 4.6 The UML StateChart for the Matchmaker

Figure 4.7 presents various classes involved during the second level matchmaking of models in terms of their tree structures. “SPProviderListGUI” calls the appropriate function in “propertiescheckbox” class to present the consumer with list of attributes of the model. The second level matchmaking constitutes calculating the “linear distance metric” and the “structure metric”. The linear distance metric between consumer tree and service provider trees is calculated by calling “comparingtrees” constructor. “SPProviderListGUI” calls the appropriate function from “structure” class and obtains the “structure metric”. These two distances are displayed to the consumer by calling the constructor of “finalresults” class. The relationships between the other classes are depicted in Figures 4 and 7 in the appendix.

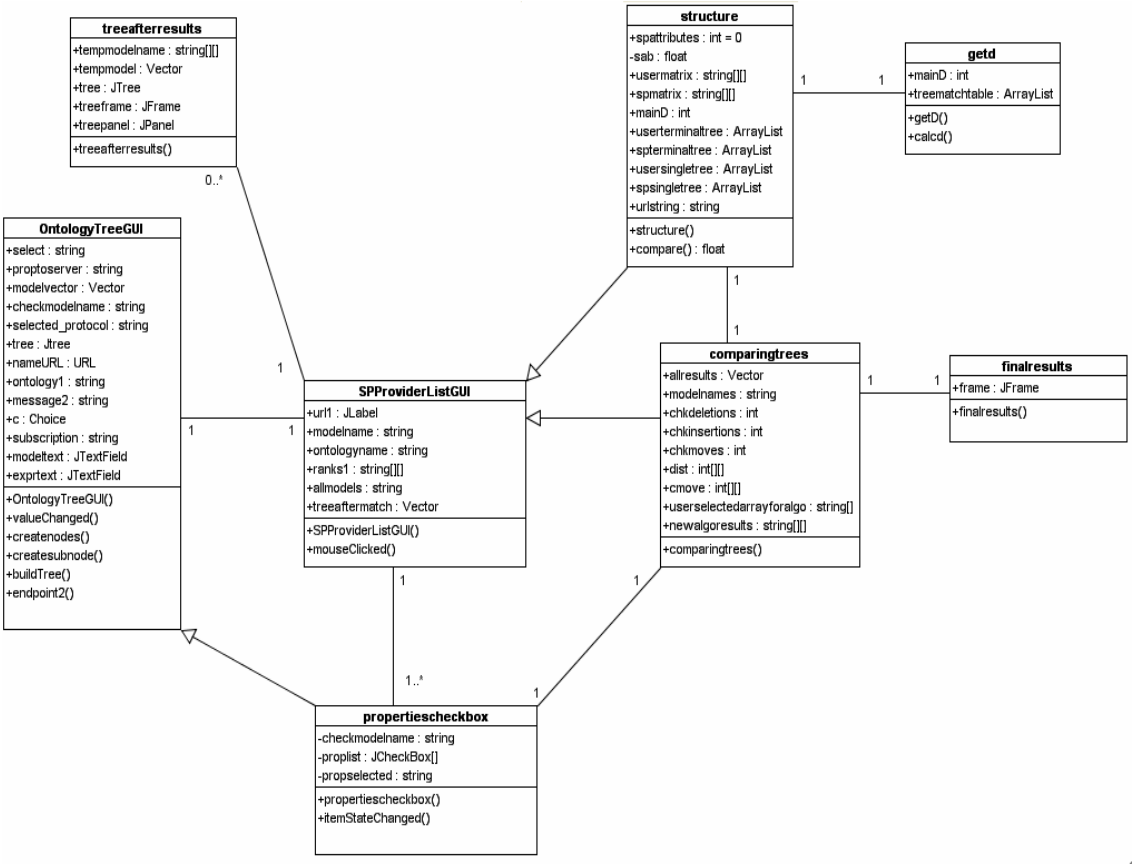

Figure 4.7 Modeling Second level Matchmaking

4.4 Service Provider Agent

The service provider agent performs the following functions:

- The service provider agent registers the service providers with the matchmaking agent.
- It also publishes the model on service provider's behalf by contacting the matchmaking agent.
    - The service provider agent first, authenticates the service provider.
    - It then takes selected ontology as input. It parses the ontology and presents the ontology tree structure to the service provider.
    - It receives the selected model and its attributes information from the service provider.

- Finally, it contacts the matchmaking agent to publish the model.

- It updates the access point of the service provider grid service.

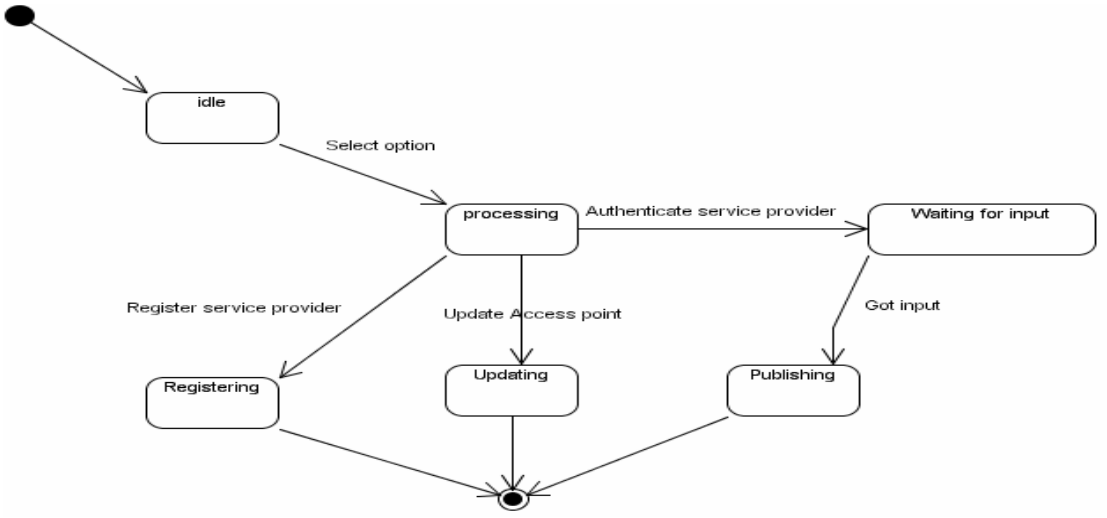

Figure 4.8 The UML State Chart for the Service Provider Agent

The various states, a service provider agent can be in are illustrated using a state diagram, in Figure 4.8. When the service provider selects an option, the service provider agent enters into "processing" state. If the service provider would like to update the access point of his grid service, the agent enters into "Updating" state. If the service provider is new and wishes to register, the agent enters into "registering" state. Finally, if the service provider wishes to publish a model, the agent authenticates the service provider, and enters into "Waiting for Input" state. Once the service provider selects the model and enters the model information, the agent enters into the "Publishing" state.

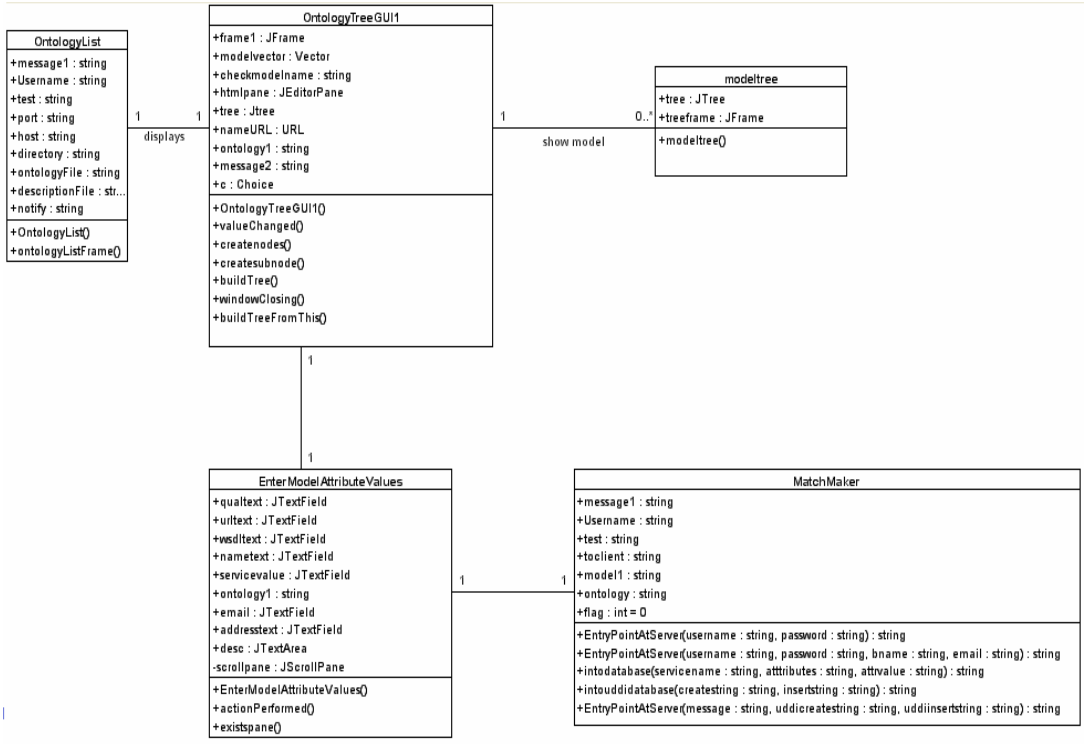

Figure 4.9 Service provider agent design class diagram

The classes that form part of the service provider agent are SPOptions, login, Register, OntologyList, OntologyTreeGUI and EnterModelAttributeValues. The relationships between these classes are shown in the class diagram.

Figure 4.9 presents the classes involved in the publish scenario. “OntologyTreeGUI” takes the service provider’s choice of the ontology from the “OntologyList” and loads the corresponding ontology. It then takes the service provider’s choice of model and its attributes using “EnterModelAttributeValues”, and passes it to the “MatchMaker”, which publishes the model into the database. “Modeltree” class visually presents the tree structure of a selected model to the service provider. The relationships between the other classes are depicted in Figures 2 and 6 in the appendix.

Chapter 5

Dynamic models of ABMS Brokering Protocols

This chapter presents the flow of logic in ABMS, using sequence diagrams. The interactions between various subsystems or components of the system are presented here.

5.1 Recommendation Protocol Scenario

Figure 5.1 visually illustrates how the interactions within ABMS take place, when the selected brokering protocol is "recommendation". The consumer interacts with the consumer agent throughout the entire process of model retrieval. The consumer agent performs all the operations on behalf of the consumer.

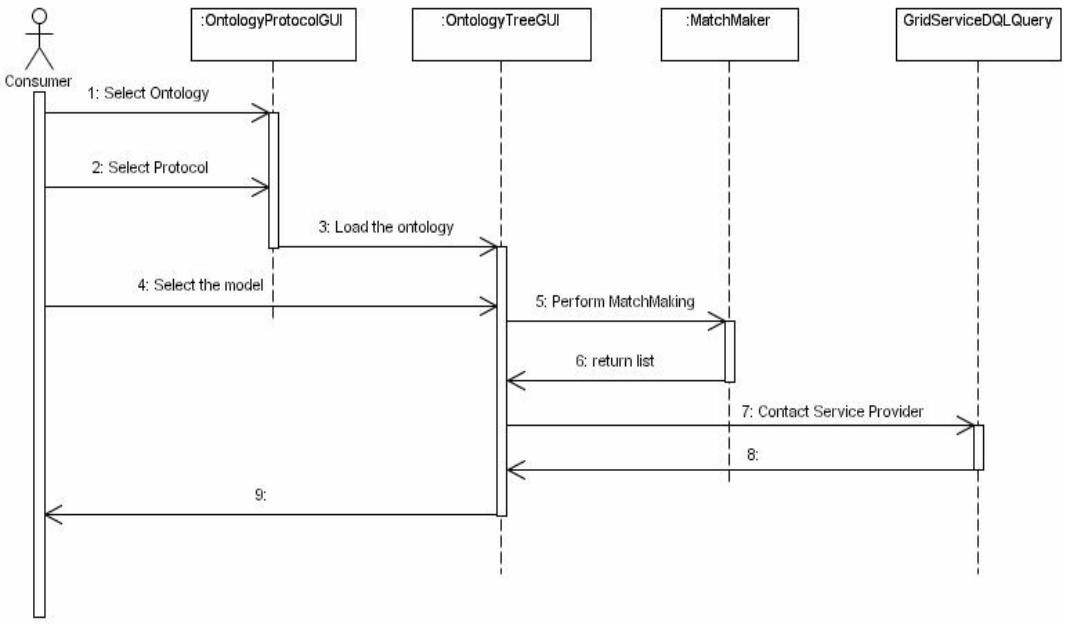

Figure 5.1. Sequence Diagram for Recommendation protocol

The classes “OntologyProtocolGUI” and “OntologyTreeGUI” form the consumer agent. In this scenario the consumer agent contacts the matchmaker agent to search and retrieve matched models. The consumer selects a service provider, then, the consumer agent contacts the brokering agent. “GridServiceDQLQuery” performs the functions of a brokering agent, contacts the service provider and retrieves model instances. The brokering agent is responsible for contacting the service provider agent in this scenario.

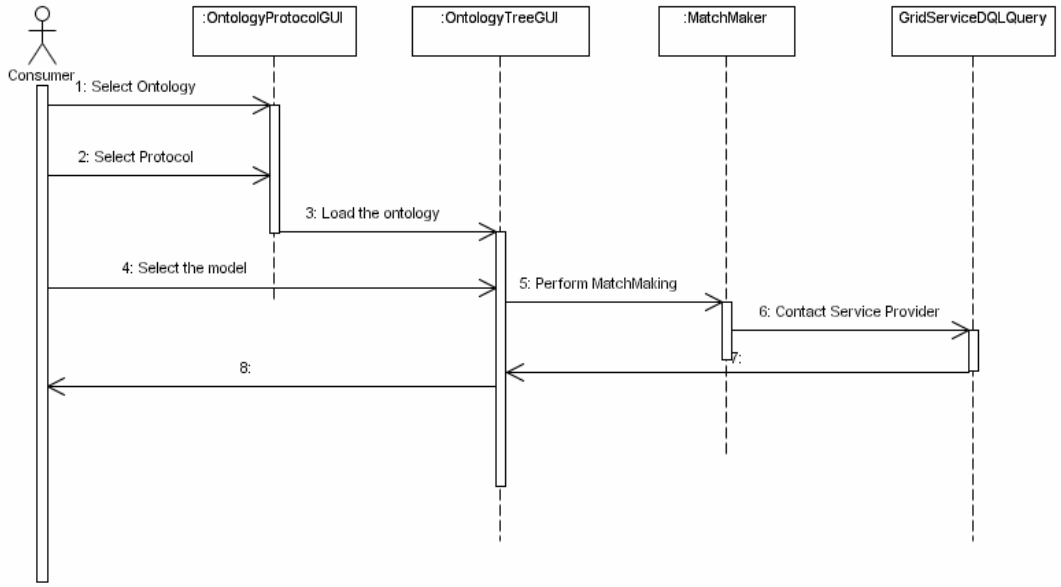

Figure 5.2. Sequence Diagram for Recruiting Protocol

5.2 The Recruit Protocol

Figure 5.2 illustrates the Recruiting protocol scenario. The recruiting protocol differs from the recommendation protocol scenario during the matchmaking process. The matchmaker agent retrieves the matched models, selects the best-matched model, and contacts the service provider agent on behalf of the consumer. The matchmaker queries the service provider agent for model instances and provides it with the consumer agent’s

address. The service provider agent then, returns the model instances back to the consumer agent.

5.3 Service Provider Agent

The service provider interacts with the service provider agent throughout the process of publishing the model, as shown in Figure 5.3. The service provider agent performs all the operations on behalf of the service provider. “Login”, “OntologyList”, “OntologyTreeGUI” and “EnterModelAttributeValues” classes form the service provider agent in the publish scenario. The service provider agent authenticates the service provider. Once authenticated, the service provider selects the ontology and the model to be published. Then it enters the model information along with attribute values. The service provider agent finally contacts the matchmaker agent, to publish the model.

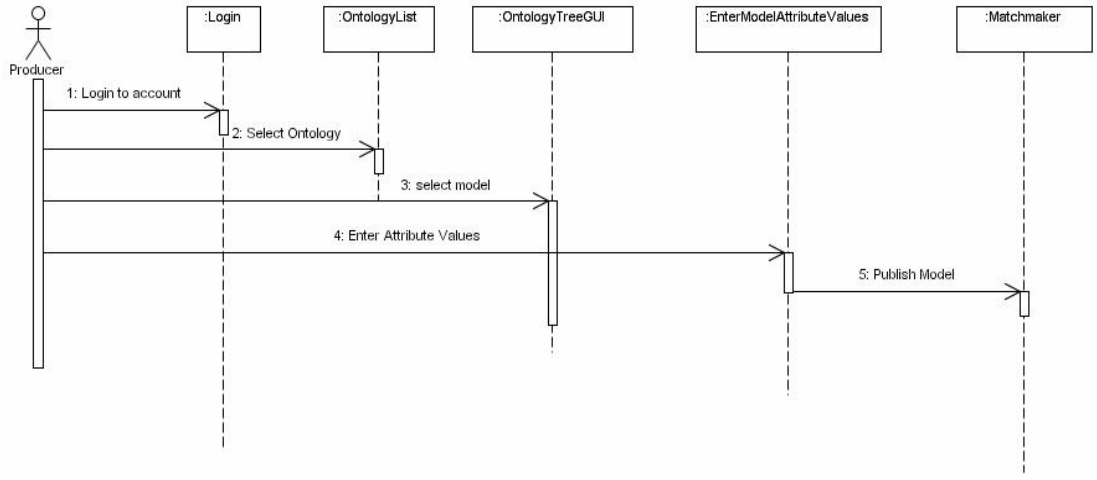

Figure 5.3. Service Provider publishing a model.

5.4 Matchmaking Scenario

Figure 5.4 illustrates the matchmaking process. The consumer agent contacts the matchmaker agent to search and retrieve models that match a consumer-selected model. The matchmaker agent performs the concept level matching and tree level matching using “Matchmaker”, “Propertiescheckbox”, “Comparingtrees” and “Structure” classes. The matchmaker performs the concept level matching and returns the matched models to the consumer agent. The consumer then selects the desired attributes of the model. The consumer agent sends the selected attributes to the matchmaker agent. The matchmaker agent then computes the linear distance metric and the structure metric for each service provider and consumer tree pair and returns the results back to the consumer agent.

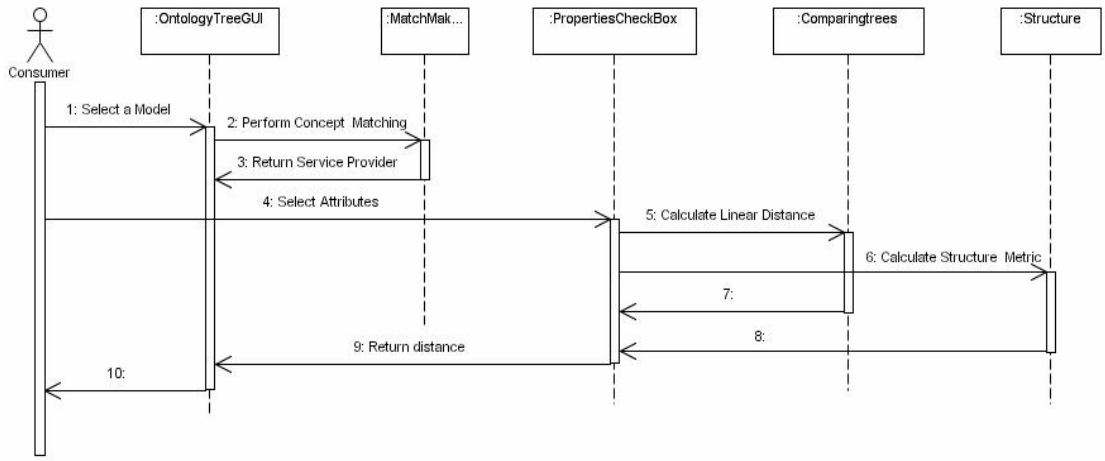

Figure 5.4. Sequence of messages during matchmaking process.

5.5 Notification of Model Availability

Figure 5.5 illustrates the notification process. The service provider agent takes input from the service provider and publishes the model by calling the matchmaker agent. The matchmaker agent illustrated by "Matchmaker", IsModelSubscribed" and "CallNotifyGridService" checks for available subscriptions and notifies the subscribed consumer agent about the availability of model.

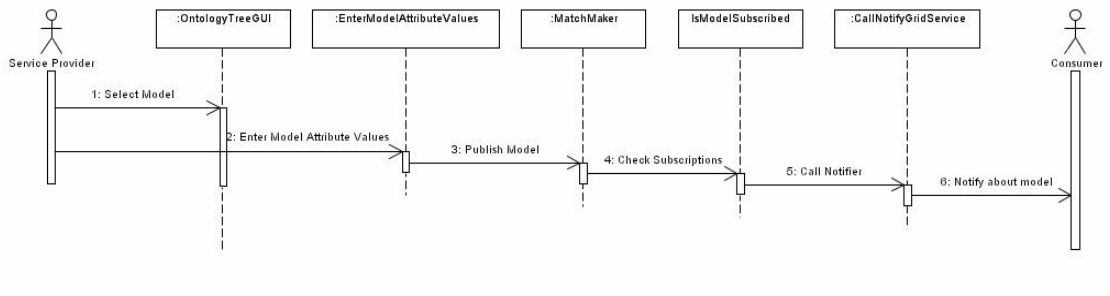

Figure 5.5. Sequence diagram modeling notification process.

Chapter 6

Implementation of the Agent-Mediated Brokering and Matchmaking System

This chapter describes the technologies and software used in implementing the ABMS. The various technologies used, grid services implemented, and the techniques used in implementing the agents are described here.

Agent-Mediated Brokering and Matchmaking System (ABMS) is a client-server application where the application logic is distributed across the server and the client applications. The underlying layer of the system is the grid, and implementation of the grid is done using Globus Toolkit 3 (GT3).

6.1 Globus Toolkit 3 (GT3)

Globus Toolkit [Borja Sotomayor 2003] is the software development environment used to program grid based applications. It provides various services, programs and utilities for the programmers. These utilities include data management, core, security, resource management and information services. In developing the ABMS, the core services of GT3 are used. The characteristics of a basic grid service are described as follows.

A grid service is exposed through its interface. An interface defines the operations and the data, the grid service provides. This interface is called portType. Any user defined grid service has to extend the standard Grid Service portType. The address of a Grid Service is a GSH or Grid Service Handle, which is similar to an URL.

Every grid service is associated with a "factory". Every time a request is made to invoke the grid service, the factory creates the grid service instance, and returns the instance to the client. A grid service provides "Service Data", a set of structured data, exposed through its interface. This Service Data provides information about the state of the service, or the information about the service itself, like cost and performance. This Service Data is essential, as it facilitates the user to select the grid service based on its characteristics. This Service Data is also used as part of the application logic, associated with the grid service. The notification facilities are also based on the changes made to the Service Data.

6.2 Technologies used

The various technologies and software that were used in building ABMS are Globus Tool Kit, Tomcat Server 3.2.3, SOAP 2.2, DAML, DQL, Java JDK, JRE 1.4.0, Jena Parser 1.6.1 and MySQL Server. The platform used is Windows.

The knowledge base for the application was built using ontologies. The knowledge base is a collection of ontologies representing the domain of military models. The military models are classified into several domains, and each domain is associated with ontology. Ontologies provide a powerful way to describe models and their relationships to other models. The DAML language developed as an extension to XML and the Resource Description Framework (RDF) is used to create Ontologies. Figure 6.1 illustrates one of the sample ontologies used in ABMS.

The latest release of the language (DAML+OIL) provides a rich set of constructs that were used to create ontologies. The ontologies are parsed using JENA parser [Jena

2004]. JENA is a parser developed to parse DAML based ontologies. The ontologies in ABMS were parsed using JENA and presented to the user as a hierarchical structure. The various models, their attributes and description information were retrieved from the ontology, and stored as temporary HTML files. The corresponding file was displayed to the user, when the model was selected from the ontology hierarchy.

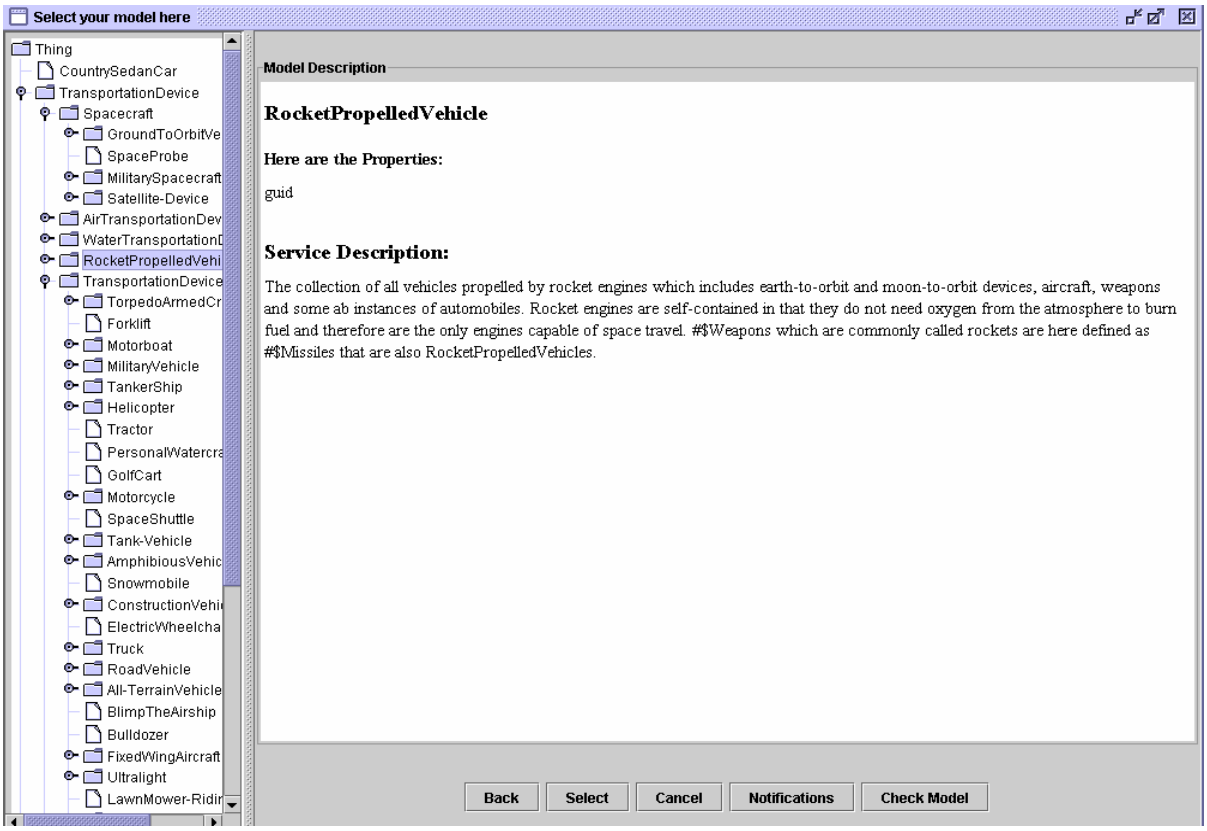

Figure 6.1. Transportation Ontology

MySQL Server was used as the advertisement database. It stores all the information related to the model advertisements, their access information as well as the subscription information. The first time a model is published by any service provider, a table is created for the model that stores all the contact information of the service

provider, as well as the model attributes information. The information for accessing the service provider grid service is stored in a separate table called “accesspt”, which will store the model name, the service provider’s GSH (Grid Service Handle) and the URL that points to the GWSDL document of the Service Provider’s Grid Service. A “subscription” table is created, that stores the information of the client that subscribed for the model. The information includes model name, ontology name, client grid service handle, and the expiry date. The client grid service handle is used while contacting the client when the desired model is available. The registration information of the service providers is stored in the “login” table.

6.3 Active Model Retrieval Mechanism

The service provider and consumer client applications, as well as the server application are deployed on the grid. The grid is installed on the server site. In addition, at every site of the consumer and the service provider, the grid is installed. Every consumer and service provider thus resides on the grid. The implementation of the consumer agent, brokering agent, service provider agent, as well as the matchmaking agent is described as follows. Three grid services are implemented in ABMS. The grid service for the service provider is implemented to access the model instances on its site. At the server site, the matchmaking agent is implemented using two grid services: one that performs matchmaking, and the other that provides notification facilities.

6.3.1 Service Provider Grid Service

The model instances database, on the service provider site, is implemented as a DAML instance file. Every model that a service provider provides contains an instance in the

database. DAML provides a set of tags and constructs that are employed, to represent each model's attribute information as an instance. The model attributes are represented as tags or elements, and the attribute values form the content for those tags. A sample model instance file for a model "US_Tank" along with its attributes like barrel, perform, speed, and others is given as follows.

```
<?xml version='1.0' encoding='UTF-8'?>

<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:vCard='http://www.daml.org/2001/03/daml+oil#'
   >

<rdf:Description rdf:about="US_Tank">
    <vCard:model>US_Tank</vCard:model>
    <vCard:barrel>yes</vCard:barrel>
    <vCard:perform>good</vCard:perform>
    <vCard:speed>50</vCard:speed>
    <vCard:ammunition_storage>yes</vCard:ammunition_storage>
    <vCard:loader_weapon>yes</vCard:loader_weapon>
    <vCard:power>5000</vCard:power>
    <vCard:coaxial_weapon>no</vCard:coaxial_weapon>
    <vCard:acceleration>450</vCard:acceleration>
    <vCard:nbc_system>yes</vCard:nbc_system>
</rdf:Description>

</rdf:RDF>
```

A grid service was implemented to access this instance database on the service provider site. Eclipse 3.0 IDE along with its Globus Toolkit plugin, is used to create this service. The first step involved in creating this grid service is writing the interface, which is a GWSDL file. The GWSDL file for the service provider grid service is as shown.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="AuburnService"
targetNamespace="http://www.globus.org/namespaces/trident/test/AuburnService"
xmlns:tns="http://www.globus.org/namespaces/trident/test/AuburnService"
xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExtensions"
xmlns:sd="http://www.gridforum.org/namespaces/2003/03/serviceData"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
xmlns="http://schemas.xmlsoap.org/wsdl/">

<import location="../../ogsi/ogsi.gwsdl"
namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
<types>
<xsd:schema
targetNamespace="http://www.globus.org/namespaces/trident/test/AuburnService"
attributeFormDefault="qualified" elementFormDefault="qualified"
xmlns="http://www.w3.org/2001/XMLSchema">
        <xsd:element name="value">
                <xsd:complexType>
                        <xsd:sequence>
                                <xsd:element name="arg1" type="xsd:string"/>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
        <xsd:element name="valueResponse">
                <xsd:complexType>
                        <xsd:sequence>
                                <xsd:element name="value" type="xsd:string"/>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>

</xsd:schema>
</types>
<message name="valueInputMessage">
        <part name="parameters" element="tns:value"/>
</message>

<message name="valueOutputMessage">
        <part name="parameters" element="tns:valueResponse"/>
</message>

<gwsdl:portType name="AuburnPortType" extends="ogsi:GridService">
        <operation name="value">
                <input message="tns:valueInputMessage"/>
                <output message="tns:valueOutputMessage"/>
                <fault name="Fault" message="ogsi:FaultMessage"/>
        </operation>

</gwsdl:portType>
</definitions>
```

The important parts of the GWSDL file are represented as bold text. The rest of the file is a generic way of creating a GWSDL file and can be ignored. The service interface is named as "AuburnPortType" and extends the simple grid service. This grid service contains an operation called "value". The parameters that the operation takes are defined using input message, and the parameter that it returns is defined using the output message. The messages further refer to the "value" and "valueResponse" elements. These elements define the number of parameters the operation takes or returns, along with their data type. In this grid service, the operation "value" takes a parameter of type "string" which is a DQL query that it obtains from the client. It uses the DQL query to query the DAML instance database, and obtains the matched model instances corresponding to the client request. These model instances are returned back to the client.

The next step is to create the namespace mappings file. To access this grid service a client requires stub classes that are generated from the GWSDL file. The namespace mappings file is used to tell the tool that creates the stub classes, where to place the stub classes. The mappings file maps the GWSDL namespaces to Java Packages, as the implementation of the service here was done in Java. The namespace mappings file is defined as follows.

```
http\://www.globus.org/namespaces/trident/test/AuburnService=trident.test.stubs
http\://www.globus.org/namespaces/trident/test/AuburnService/bindings=trident.test.stub
s.bindings
http\://www.globus.org/namespaces/trident/test/AuburnService/service=trident.test.stubs.
service
```

Next we discuss the implementation of the service. The implementation is comprised of a java class that implements the operations defined in the GWSDL file.

This service is deployed onto the grid using a deployment descriptor. The deployment descriptor was written in WSDD (Web Service Deployment Descriptor) format. The deployment descriptor specifies the path to the grid service. The base address of our grid service container combined with this path, forms the address or GSH of the service provider grid service. The three files were combined to a single GAR file "trident_test.gar" using Ant. Ant is a java build tool. It takes the three files and creates an executable GAR file out of it. This GAR file is then deployed onto the grid using ant deploy command as follows.

ant deploy -Dgar.name=sample\build\lib\trident_test.gar

sample\build\lib is the path where "trident_test.gar" grid service file resides.

6.3.2 Matchmaking Grid Services

The server hosts two grid services that provides matchmaking as well as notification facilities. The creation of matchmaking grid service is similar to the one described above. The notification grid service differs in that, in addition to the simple grid service portType, it also extends the notification portType to support notification facility.

6.3.3 Agents

The consumer, service provider and brokering agents are implemented using Java. The consumer and the service provider agent applications access the knowledge database of ontologies from the server, parse them using JENA, and then present the concept hierarchy to the users.

6.3.3.1 Consumer Client Application

The consumer client begins with the list of ontologies available at the server side. Figure 6.2 shows the initial GUI. The GUI presents two tabs, the ontology tab and the protocol tab. The consumer begins by selecting one of the ontologies and the protocol. The description of each of the protocol and the ontology is shown when it is selected.

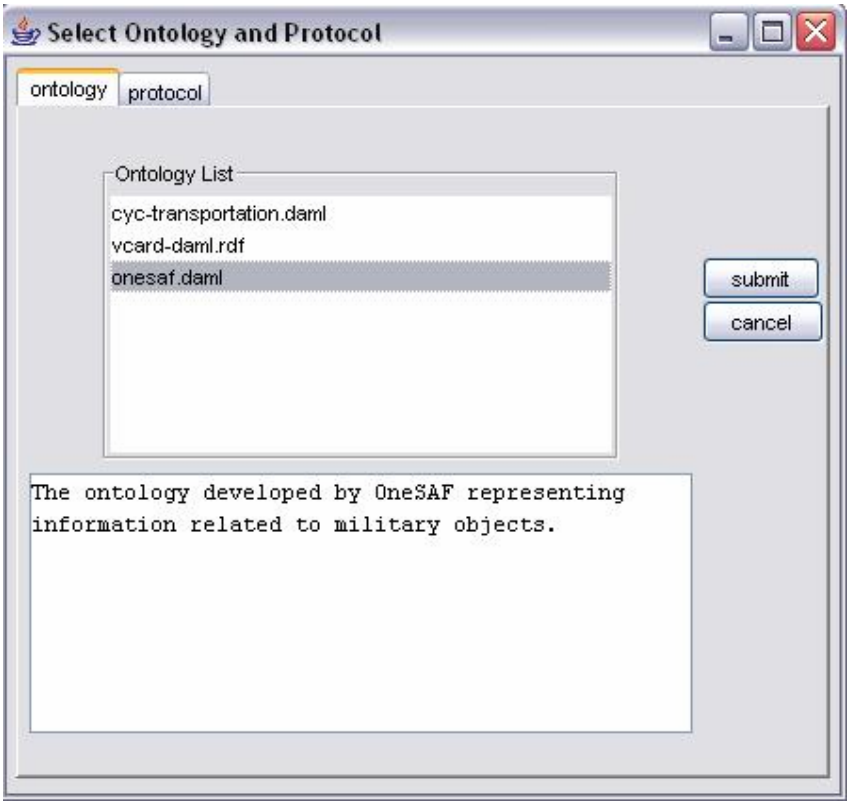

Figure 6.2. Consumer Initial Interface Pane

The consumer agent then retrieves the ontology file from the server, parses it, and presents the ontology tree to the consumer. The tree hierarchy is similar to one shown in Figure 6.1. The ontology pane contains the tree onto the left, and an editor pane onto the right. The editor pane loads the description and other information of the model, when the

model is selected on the tree, on the left. The notification button shows any notifications currently available for the consumer. The consumer client application also prompts the consumer, of any available notifications, when the application is loaded.

After the consumer makes a selection of the model, the consumer agent makes a request to the server, for models that are available. The server processes the request, and returns the models that are matched. Figure 6.3 shows the interface that shows the available models.

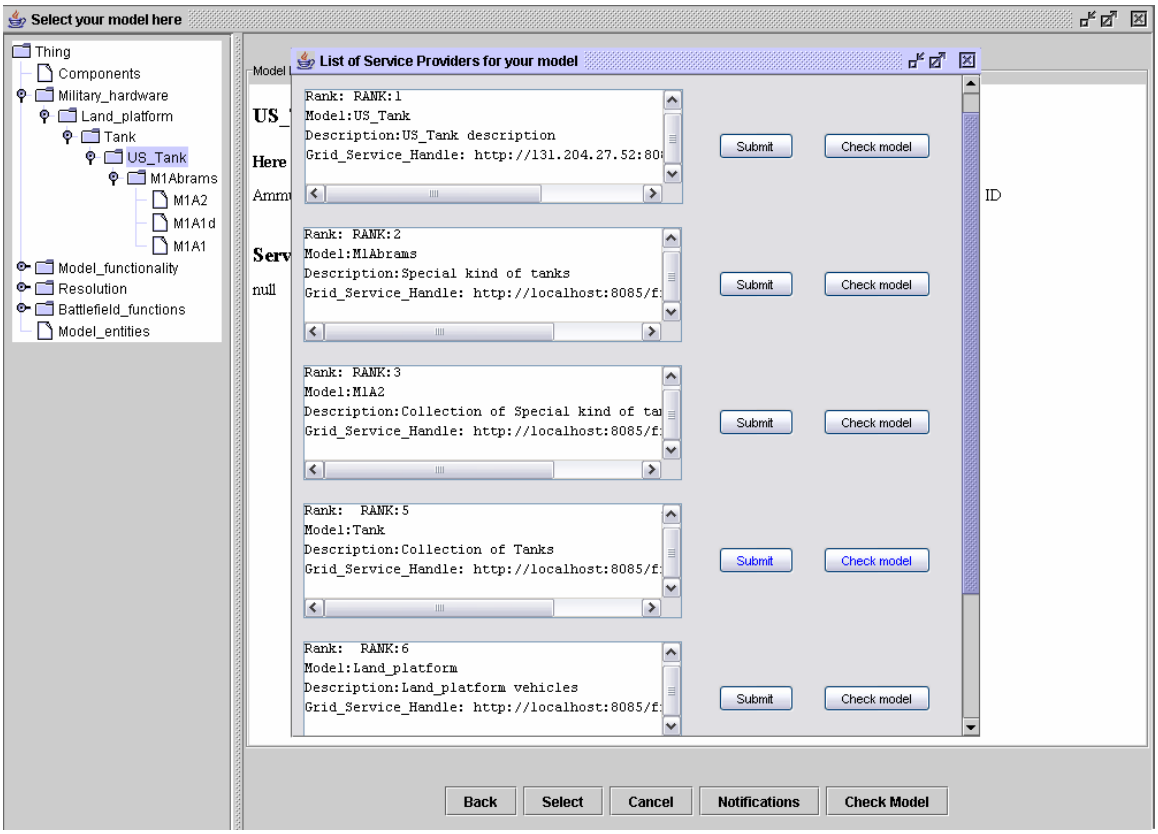

Figure 6.3. List of matched models

In the figure, the model “US_Tank” was searched for. All the models that matched were returned. As seen in the ontology, all the available models that were at the

lower and higher levels of "US_Tank" were returned, with their corresponding ranks. The models at the lower level represent more specialized models, and hence are ranked higher than the ones, which are at higher level than "US_Tank". Among the models at lower level, the ones which are closer to the selected model "US_Tank" are ranked and recommended as closely matched ones, than the others. Similar approach is employed for models at higher levels.

If the selected protocol is recommendation, the matchmaker returns the list of matched models. The user next, makes a selection of set of attributes, he desires. The structure and linear distance metrics are calculated and presented to the user.

In Figure 6.3, some of the models that partially matched the selected model, during first level of matchmaking are "US_Tank", "M1Abrams" and "Tank" models. Based on the attributes chosen by the service provider, the tree structures associated with these models are shown in Figure 6.4.

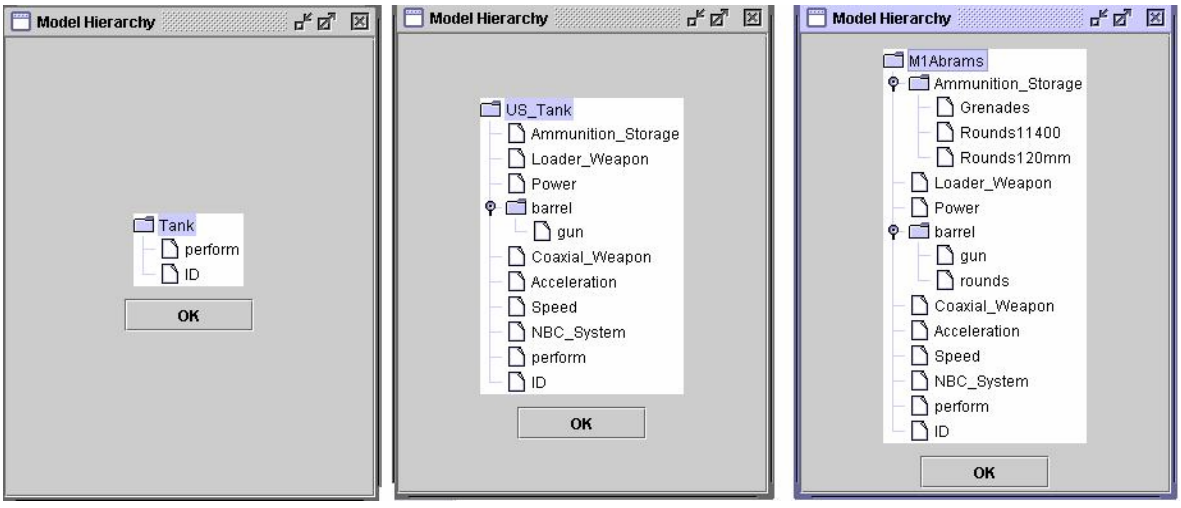

Figure 6.4 Tree Structures of Tank, US_Tank and M1Abrams

The attributes of the model, selected by the consumer are Ammunition_Storage, Rounds120mm, barrel, rounds and Speed. The tree for the consumer model is shown in Figure 6.5. From Figure 6.4 and 6.5, it is clearly seen that the consumer tree best matches the model “M1Abrams”, next it best matches “US_Tank” and the last is “Tank”. This is supported by the distance metrics presented in Figure 6.6.

The consumer next selects the desired model, and the brokering agent calls the service provider to query its model instances database. The consumer enters the values of the attributes; the brokering agent forms a DQL query of the attributes, as well as values and sends them to the service provider grid service. The grid service queries its instance database, and retrieves the models that match the attribute values. The model instances are returned back to the consumer.

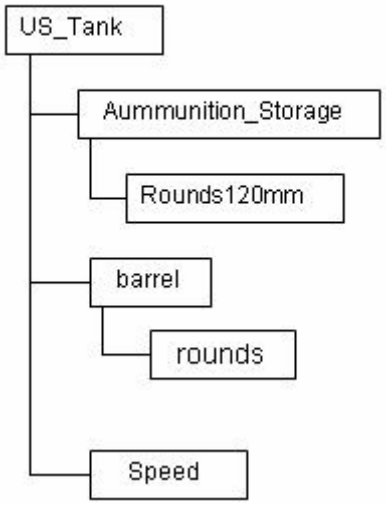

Figure 6.5 Consumer tree

In the notification scenario, the consumer selects the model he is interested in subscribing to. The consumer enters the expiry date for the subscription. When the model is available, the notification grid service is called and the consumer is notified, about the availability of the model. Figures 6.7 and 6.8 depict the notification scenario.

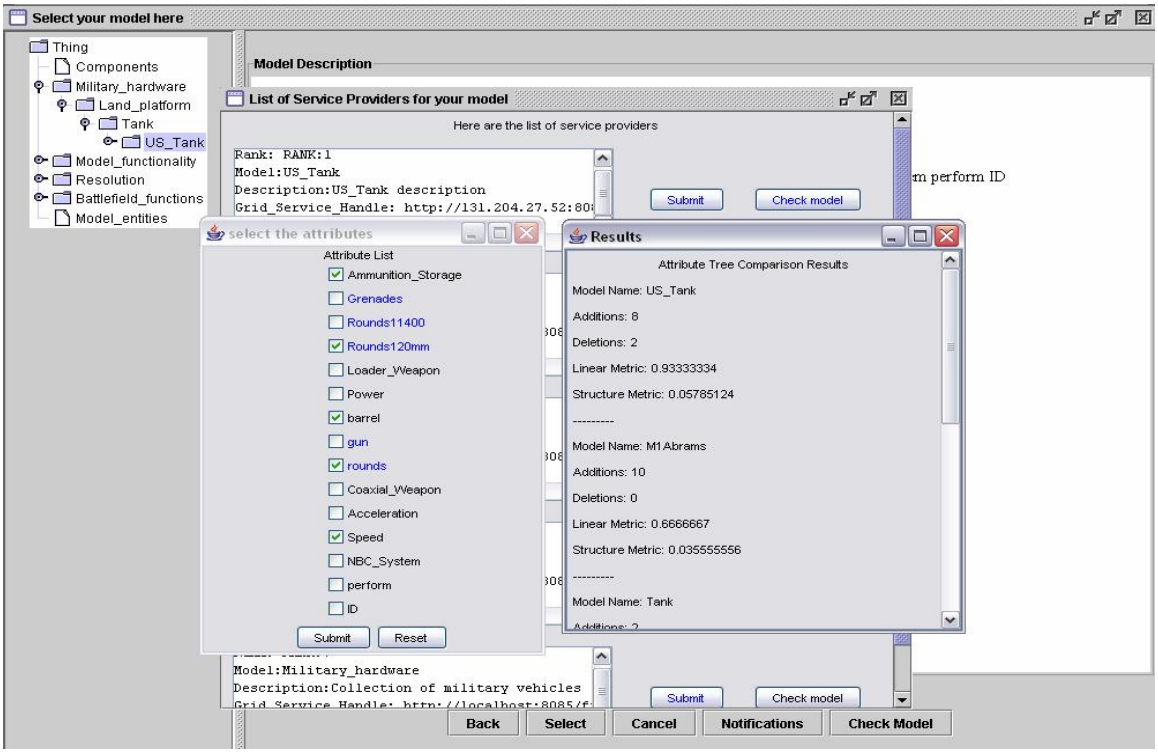

Figure 6.6. Tree level Matchmaking

6.3.3.2 Service Provider Client Application

The service provider's agent application begins with a list of options. The options include logging into the system before publishing a model, registering if it's a new service provider, and updating the access point of the grid service. The service provider logs on to his account to publish a model. After logging into the system, he is provided with a list of ontologies, from which he makes a choice as shown in Figure 6.9. The ontology is

parsed and presented to the user as a tree structure. The service provider chooses from various model classifications in the ontology, to publish his model, as shown in Figure 6.10.

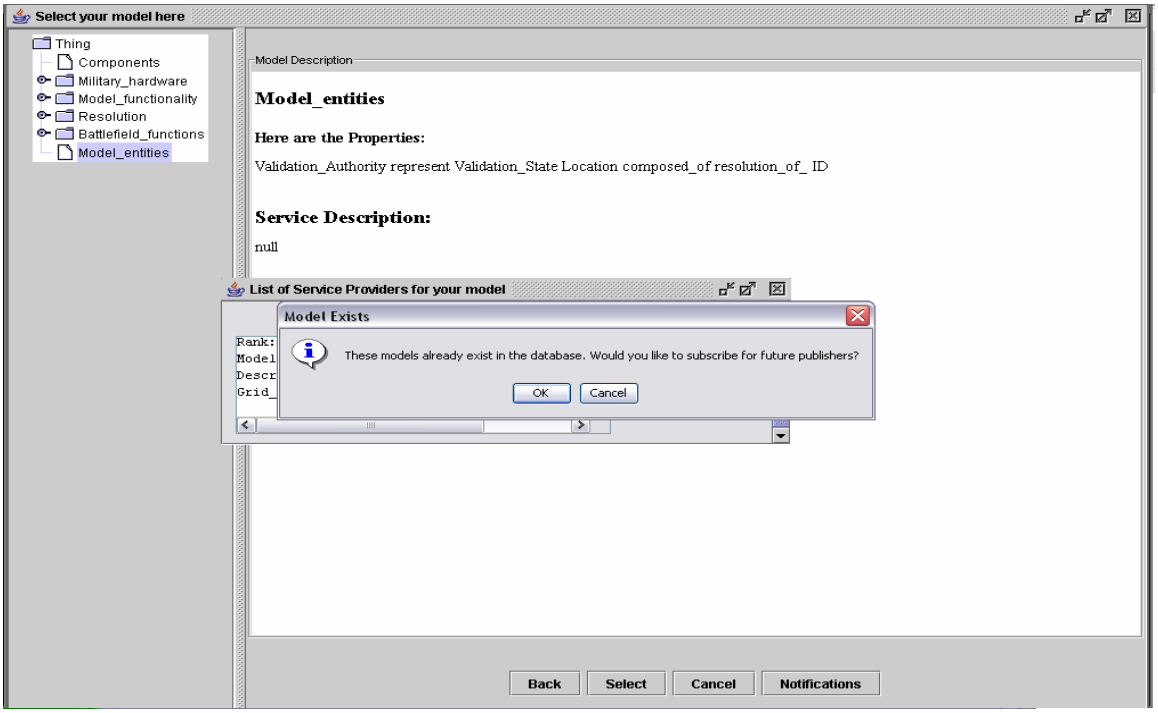

Figure 6.7 Subscription Scenario

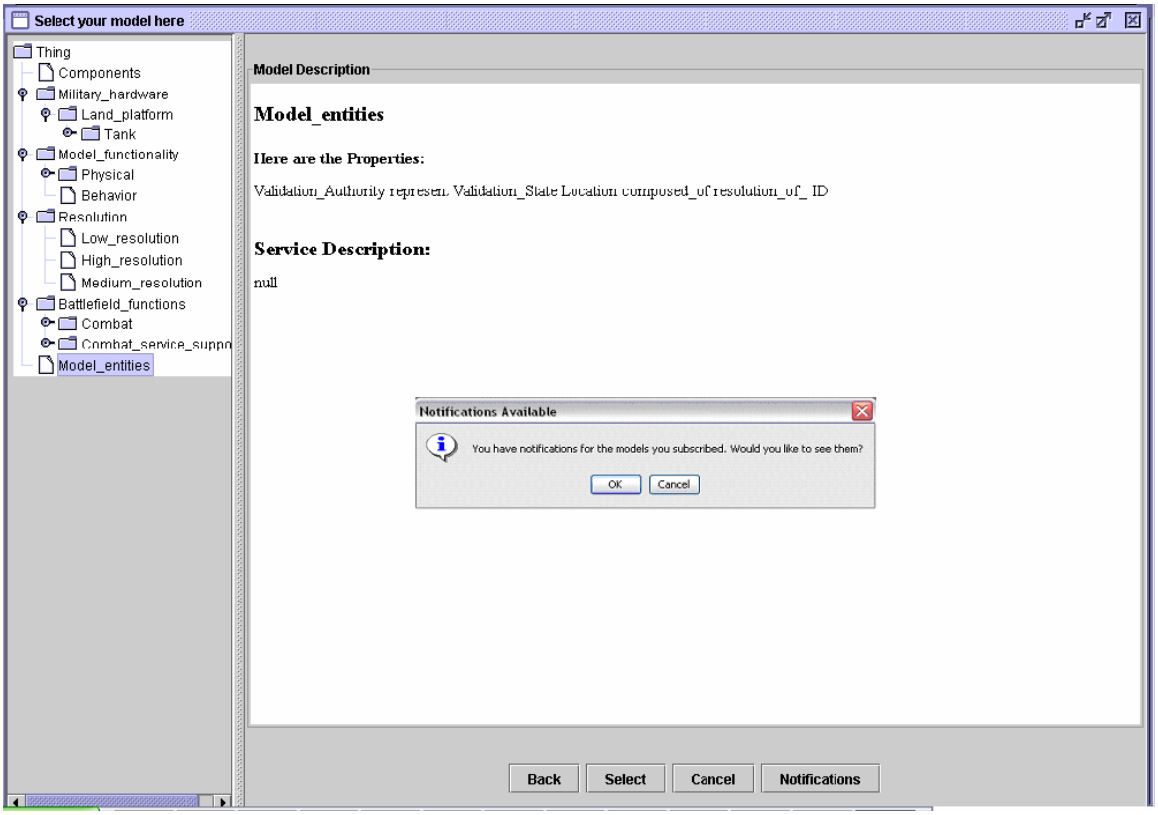

Figure 6.8 Consumer notified about the available models.

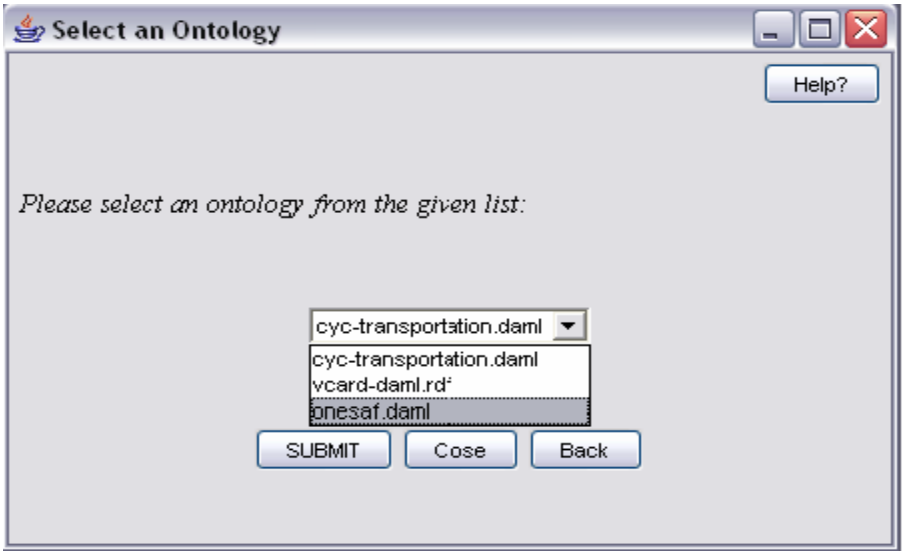

Figure 6.9 Selecting an Ontology

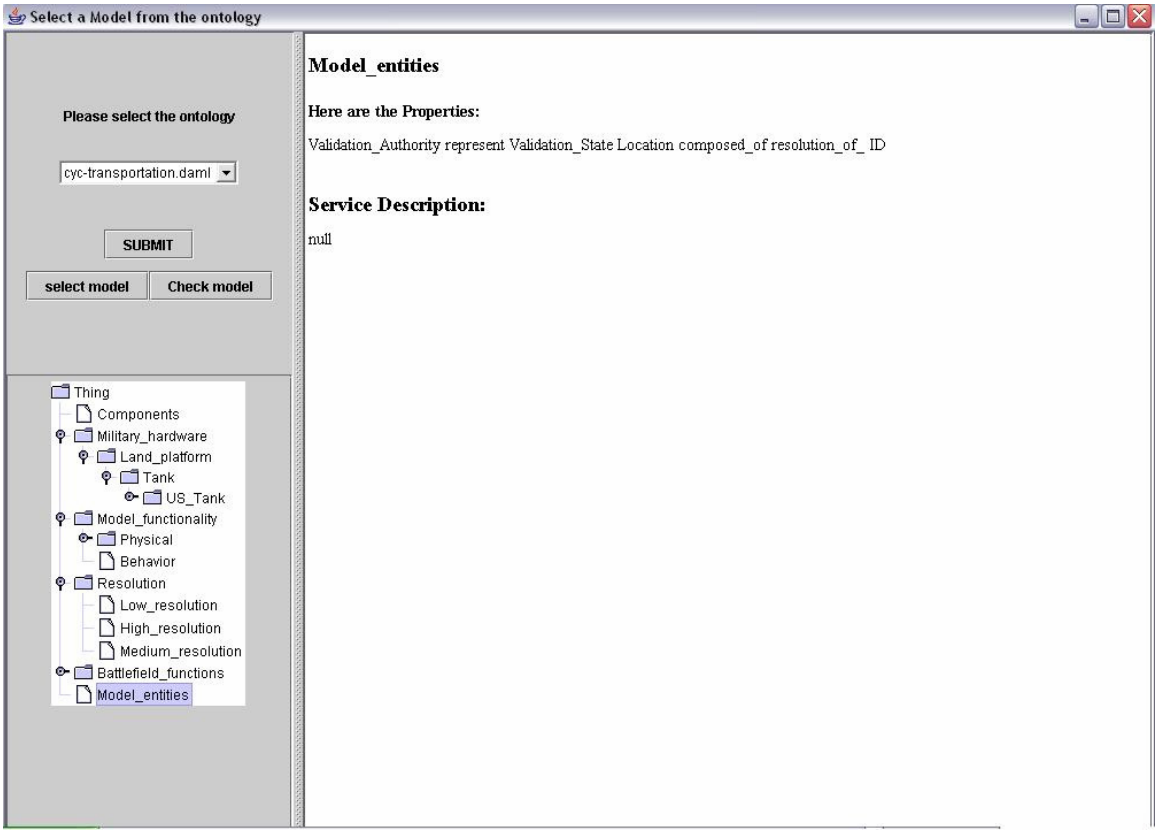

Figure 6.10 Selecting a model classification from the ontology

The service provider next enters the model attribute information along with his contact information. He enters the attribute values of the model. The grid service handle or the grid service URL is entered as "http://localhost:8082/ogsa/services/trident/test" and the URL of the interface file of the grid service is entered as "http://localhost: 8085/files/Auburn.gwsdl".

Once the information is entered, the service provider agent contacts the matchmaker agent that publishes the model, into the advertisement database. The service provider is informed when the model is successfully published.

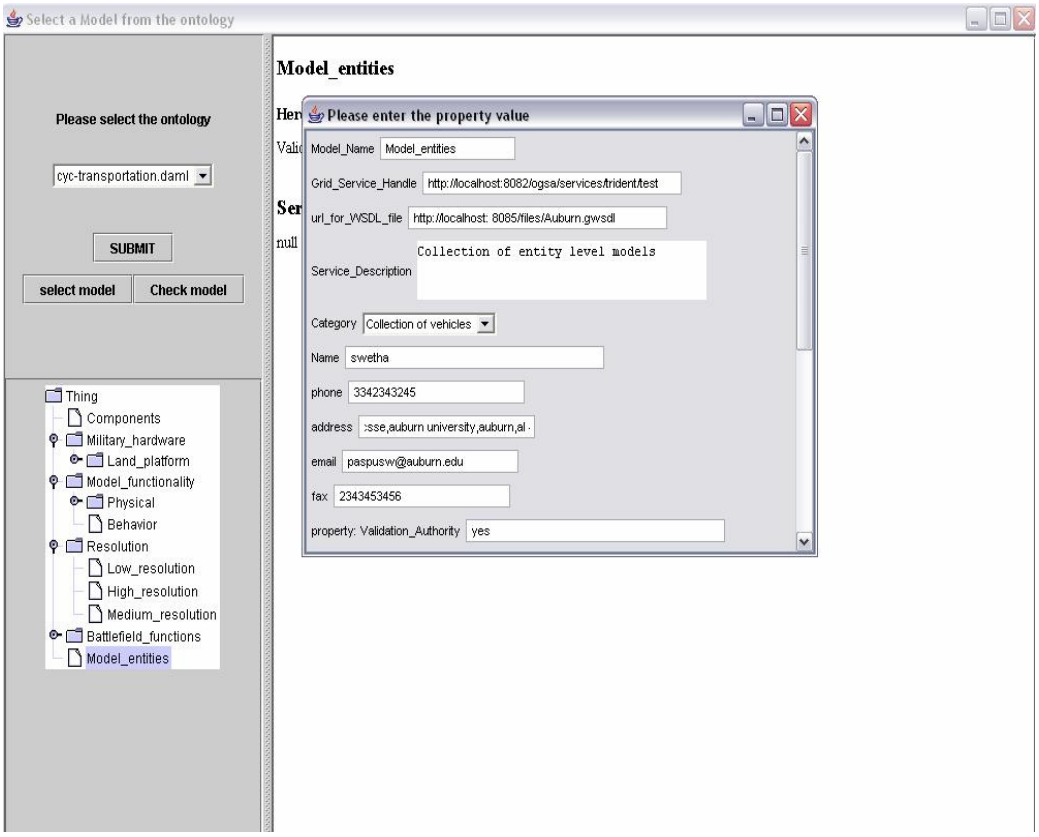

Figure 6.11 Entering the attribute values of the model.

6.4 Limitations

The current approach used in implementing the ABMS has the following limitations. The grid services created are unsecured. These services need to be made secure, to facilitate authenticated and confidential communications between the grid service, and the client invoking it. These grid services can be made secure by using the security services provided by the GT3. Use of certificates provides secure communication between the grid elements.

The model instances at the service provider were created manually. A GUI tool that automatically creates the model instances, upon input from the service provider is desired. Also the service provider has to remember the technical aspects of the grid service, like the grid service handle address, and the location of the grid service interface file. The current implementation requires that he has knowledge of this information while publishing a model. It would be a feasible and optimal solution to relieve the service provider from this information by some means, like hard coding the addresses.

The current approach is also aimed at discovering and locating the suitable model providers. How the negotiations between the consumer and the selected service provider are performed, for acquiring the model, is not addressed here. The transactions are addressed using negotiation protocols which are discussed in the next chapter.

Chapter 7
Conclusions and Future Work

In this thesis, an agent-based active model retrieval mechanism was developed and implemented. The system was tested with various test cases to ensure that the system works exactly as intended. The various brokering protocols and the matchmaking mechanism, along with the publishing scenario were tested with several inputs. The matched models results obtained from the matchmaking algorithm provided accurate results. When a consumer requested model was not available, the first phase of filtering employed here returned models that partially matched, along with their degree of match. The computed degree of similarity provided satisfactory results.

The system was also tested for the subscription and notification scenarios. The system delivered appropriate notification messages when a model became available. The system also delivered notifications when the published model was a partial match to the subscription. All the test cases ensured that the system functions properly as desired.

The mechanism addressed various limitations introduced by the conventional methods. The potential benefits of the system are that the system facilitates reusability of the simulated models. The developers are relieved from the effort to create a simulated model from scratch every time it is desired. Reusability improves project efficiency and reduces effort.

The mechanism facilitates new service providers to make contributions by publishing and promoting their simulated models. In addition, the mechanism provides a means for the consumer to subscribe for future models and be notified of their availability. This relieves the users from constantly interacting with the system and searching for desired models. This approach of combination of grid infrastructure and agent-based matchmaking and brokering schemes can be used in various projects, in e-commerce and e-services like distributed virtual workgroups or virtual corporations, where several businesses interact with each other for information discovery, sharing, and retrieval.

The underlying layer for the Agent-Mediated Brokering and Matchmaking System is the grid. The grid performs the location and retrieval of models, as well as the notification/subscription facilities. The next layer built on the top of this layer, the intelligent services layer, performs the matchmaking and brokering functions through the use of agents.

The system performs the discovery of models by developing a partial matchmaking mechanism. A two-phase filtering approach is employed in matchmaking, and the models that conceptually and structurally match the desired model are retrieved. The degree of similarity of each model to the intended or desired model is also computed to rank qualified models.

The system also presents a set of brokering protocols for contacting the service provider. These brokering modes facilitate transparency between the consumers and the

service providers. The brokering modes include Recommendation, Recruiting and Notification/ Subscription.

In this thesis, the attributes of the models are listed in terms of a textual representation. This approach can be improved by presenting the model tree structure to the consumer in a graphical and interactive way. The trees will denote concept hierarchies in which the nodes represent the properties of the concept. When the consumer would like to delete an attribute, he can select the corresponding node and delete the attribute. Similarly, when inserting an attribute he can select a node and insert the attribute as the child for the node. This improves the user interface of the system, provides flexibility to the consumer, and avoids confusion. Similar improvement can be achieved for the service provider.

This thesis is aimed at discovering and locating the simulated models, and contacting the service provider for the model instances. The transactions and potential negotiation mechanisms for acquiring qualified and selected models are not addressed in this thesis. Incorporating negotiation protocols into the system is left as a future work.

Negotiation [Guttman et al. 1998] is the process of agreeing on the terms of the contract for performing transactions between the consumer and service provider. Typical examples of terms of contract include price, warranty, date of delivery etc. The entities involved in the negotiation process come to an agreement on the terms of contract like setting the price or the delivery time. The negotiation process generally lasts for several iterations until either a decision is reached, or the process is terminated [Lee 2004]. In each iteration, an entity proposes an offer based on the terms, and the opposite entity

responds either by accepting it or making a counter offer. Each entity uses its own strategy in computing the offer [Morris et al. 2000].

In ABMS, the negotiation can be performed using agent techniques. Two agents called buying agent [Morris et al. 2000] and selling agent [Morris et al. 2000] can be used in the negotiation process. The consumer may interact with the buying agent and actively involve in the negotiation process, and the service provider may interact with the selling agent to participate in the negotiation process. These agents make offers to each other throughout the negotiation process. The scenario involved in this system is a business to consumer scenario, where a consumer interacts with a service provider. The ratio of number of consumers to service providers is 1:1 although the negotiation takes place for several models provided by the service provider. Hence the consumer to service provider ratio would be 1: m, where m is the number of models available at service provider site. The terms involved in negotiation can be price and delivery time. These constraints are fuzzy [Kurbel and Loutchko. 2003], that is, the consumer or the service provider specifies a range of values for the terms. The buying agent and the selling agent use their own strategies to make the offers. The negotiations can be done for three levels. The process ends when both reach an acceptable price and delivery date, or after three levels when the negotiation is terminated, without reaching any acceptable price and delivery time.

References

Borja Sotomayor 2003. The Globus Toolkit 3 Programmer's Tutorial. http://gdp.globus.org/gt3-tutorial/singlehtml/progtutorial_0.4.3.html

CAE 2004. "Modeling and Simulation Challenges and CAE's Capabilities." http://www.cae.com/www2004/Products_and_Services/Military_Simulation_and_Training/Brochures/Modeling_Simulation_Solutions.pdf

Daniel Kuokka and Larry Harada. 1995. "Matchmaking for information agents." In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pages 672-678, Montr'eal, Canada, August 1995.

Decker, k., M. Williamson, and K. Sycara. 1996. "Matchmaking and brokering." In Proceedings of the 2nd Intl. Conf. on Multi-Agent Systems, 1996.

Durfee, E.H., D. L. Kiskis, and W. P. Birmingham. 1997. "The agent architecture of the university of michigan digital library." IEE Proc on Software Engineering, 144(1):61--71, 1997.

Fensel, D. 1997. "An ontology-based broker: making problem-solving method reuse work". In Proceedings of the Workshop on Problem-Solving Methods for Knowledge-based Systems (W26) held in conjunction with the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97), pp. 23--29, Japan, August 1997.

Finin T., J. Weber, G. Wiederhold, M. Genesereth, D. McKay, R. Fritzson, J. McGuire, R. Pelavin, S. Shapiro and C. Beck. 1993. "Draft specification of the KQML agent communication language". Technical Report, the ARPA Knowledge Sharing Initiative External Interfaces Working Group, 1993.

Foner, L.N., 1993. "What's an agent anyway: A sociological case study." Agents Memo 93-01,Agents Group, MIT Media Lab, 1993.

Foster, I. and Kesselman, C. 1999. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, 1999

Foster I., C. Kesselman, S. Tuecke. 2001. "The Anatomy of the Grid – Enabling Scalable Virtual Organizations." International J. Supercomputer Applications, 15(3), 2001.

Foster, I., Kesselman, C., Nick, J.M. and Tuecke, S. 2002. "Grid Services for Distributed Systems Integration." IEEE Computer, 35 (6). 37-46. 2002.

Foster, C. Kesselman, J. Nick, and S. Tuecke. 2002. "The physiology of the grid: An open grid services architecture for distributed systems integration." http://www.globus.org/research/ papers/ogsa.pdf, June 2002.

Franklin, Stan and Graesser, Art. 1997. "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents," Proceedings of the Agent Theories, Architectures, and Languages Workshop, Berlin: Springer Verlag, 193-206.

Genesereth, M.R., S.P. Ketchpel. 1994. "Software Agents." Communications of the ACM, volume 37, Issue 7, Pages 48-55, July 1994.

Guttman R., A. Moukas, and P. Maes. 1998. *Agents as Mediators in Electronic Commerce.* International Journal of Electronic Markets*,* Vol. 8, No. 1, February 1998.

Hayden S., C. Carrick, and Q. Yang. 1999. "Architectural Design Patterns for Multiagent Coordination". In Proc. of the 3rd Int. Conf. on Autonomous Agents, Agents'99, Seattle, USA, May 1999.

IBM Redbooks. 2004. *Grid Services Programming and Application Enablement.* May 2004.

Jena 2004. http://jena.sourceforge.net/tutorial/RDF_API/index.html

Katia P. Sycara, Joseph A. Giampapa, Brent Langley, Massimo Paolucci. 2002. "The RETSINA MAS, a Case Study.", Pages 232-250, SELMAS 2002.

Ken Baclawski, 1998. http://www.ccs.neu.edu/home/kenb/com3337/rmi_tut.html

Kurbel, K., Loutchko I. *Towards Multi-Agent Electronic Marketplaces: What is there and What is Missing?*. The Knowledge Engineering Review. 18 (1), pp. 33-46, 2003.

Levent Yilmaz. 2002. "Towards a Metric Suite for Discrete Event Trace Validity. Simulation and Software Division". Trident Systems Incorporated, VA, 2002.

Liophant. 2003. "Combat Vehicle Simulator." http://www.liophant.org/liophant/projects/combatsim.pdf

Ludwig and Santen. 2002. "A Grid Service Discovery Matchmaker based on Ontology Description" http://datatag.web.cern.ch/datatag/papers/euroweb2002.pdf

M. Genesereth. 1992. "An agent-based framework for software interoperability." In Proceedings DARPA Software Technology Conference, 1992.

Michael Weiss. 1998. "A gentle introduction to agents and their applications". MITEL Corp, 1998. http://www.magma.ca/~mrw/agents/what.html

Morris J., P. Ree, and P. Maes. 2000. "Sardine: Dynamic seller strategies in an auction marketplace". In Proceedings of the Second ACM Conference on Electronic Commerce (EC-00), pages 128-134, 2000.

Mühl G., L. Fiege, A. Buchmann. 2002. "Filter Similarities in Content-Based Publish/Subscribe Systems". International Conference on Architecture of Computing Systems, (ARCS'02).

Murphy, J. P., G. A. Mills-Tettey, L. F. Wilson, G. Johnston, and B. Xie. 2003. "Demonstrating the ABELS system using real-world scenarios." In Proceedings of the 2003 SAINT Conference, pages 74-83, January 2003.

Nwana, H.S., 1996. *Software Agents: An Overview*. Knowledge Engineering Review, Vol. 11, No 3, pp.1-40, Sept 1996.

OneSAF. 1999. "One Semi-Automated Forces (OneSAF) Mission Needs Statement (MNS)", August 1999.

Page, E.H., Roger Smith. 1998. "Introduction to Military Training Simulation: A Guide for Discrete Event Simulationists." Proceedings of the 1998 Winter Simulation Conference. December 1998.

Paolucci M., Takahiro K., Terry, R.P., K.P. Sycara. 2002. "Semantic Matching of Web Service Capabilities." In Proceedings of the First International Semantic Web Conference on The Semantic Web, pages 333-347, Sardinia, Italy, June 2002.

Retsina. 2001. CMU's Retsina agent architecture Web site, 2001: http://www-2.cs.cmu.edu/~softagents/retsina_agent_arch.html

Richard E. Hayes. 2002. “The Future of Military Modeling and Simulation, As Seen Through the Eyes of the Military Operations Research Society Membership.” The Proceedings of the 2002 Summer Computer Simulation Conference, 2002.

Romanowski, C.J. and Nagi, R. 2005. "On Comparing Bills of Materials: A Similarity/Distance Measure for Unordered Trees". IEEE Transactions on Systems, Man, and Cybernetics, Part A, Vol. 35(2), pp. 249-260, 2005.

Savas Parastatidis. 2003. http://www.neresc.ac.uk/events/presentations-talks/WebGridServices.pdf

Singh, N. 1993. “A Common Lisp API and Facilitator for ABSI: Version 2.0.3”. Technical Report logic-93-4, Logic Group, Computer Science Department, Stanford University, 1993.

Sycara K., Seth Widoff, Matthias Klusch and Jianguo Lu. 2002. “LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace.” Autonomous Agents and Multi-Agent Systems, 5, 173–203, 2002.

Tolk, A., 2004. “Composable Mission Spaces and M&S Repositories-Applicability of Open Standards.” 2004 Spring simulation Interoperability Workshop. April 2004.

Tom Gruber. 1993. “What is an Ontology”. http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

W3C., 2000. Simple Object Access Protocol (SOAP) 1.1. W3C, Note 8, 2000.

Wallace Croft, David. 1997. “Intelligent Software Agents: Definitions and Applications”. 1997. http://www.alumni.caltech.edu/croft/research/agent/definition.

Wei-Po Lee. 2004. *Towards agent-based decision making in the electronic marketplace: interactive recommendation and automated negotiation.* Expert Systems with Applications , Volume 27, Number 4, pages 665-679, November 2004.

Whittman, R. and C. Harrison. 2001. “OneSAF: A Product Line Approach to Simulation Development.” In Proceedings of the European Simulation Interoperability Workshop, June 25, 2001.

Wooldridge, Michael and Nicholas R. Jennings. 1995. “Agent Theories, Architectures, and Languages: a Survey”, in Wooldridge and Jennings Eds., Intelligent Agents, pages 1-22, Berlin: Springer-Verlag, 1995.

Zeigler, B., 2000. “Model Repositories and Assembly and Integration of Models.” Technology for the United States Navy and Marine Corps, 2000-2035 Becoming a 21st-Century Force, Volume 9, Appendix F, 2000.

Zhang, Z. and Zhang, C. 2002. “An Improvement to Matchmaking Algorithms for Middle Agents.” Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, pages 1340-1347, The Association for Computing Machinery, NY – USA.

Appendix A

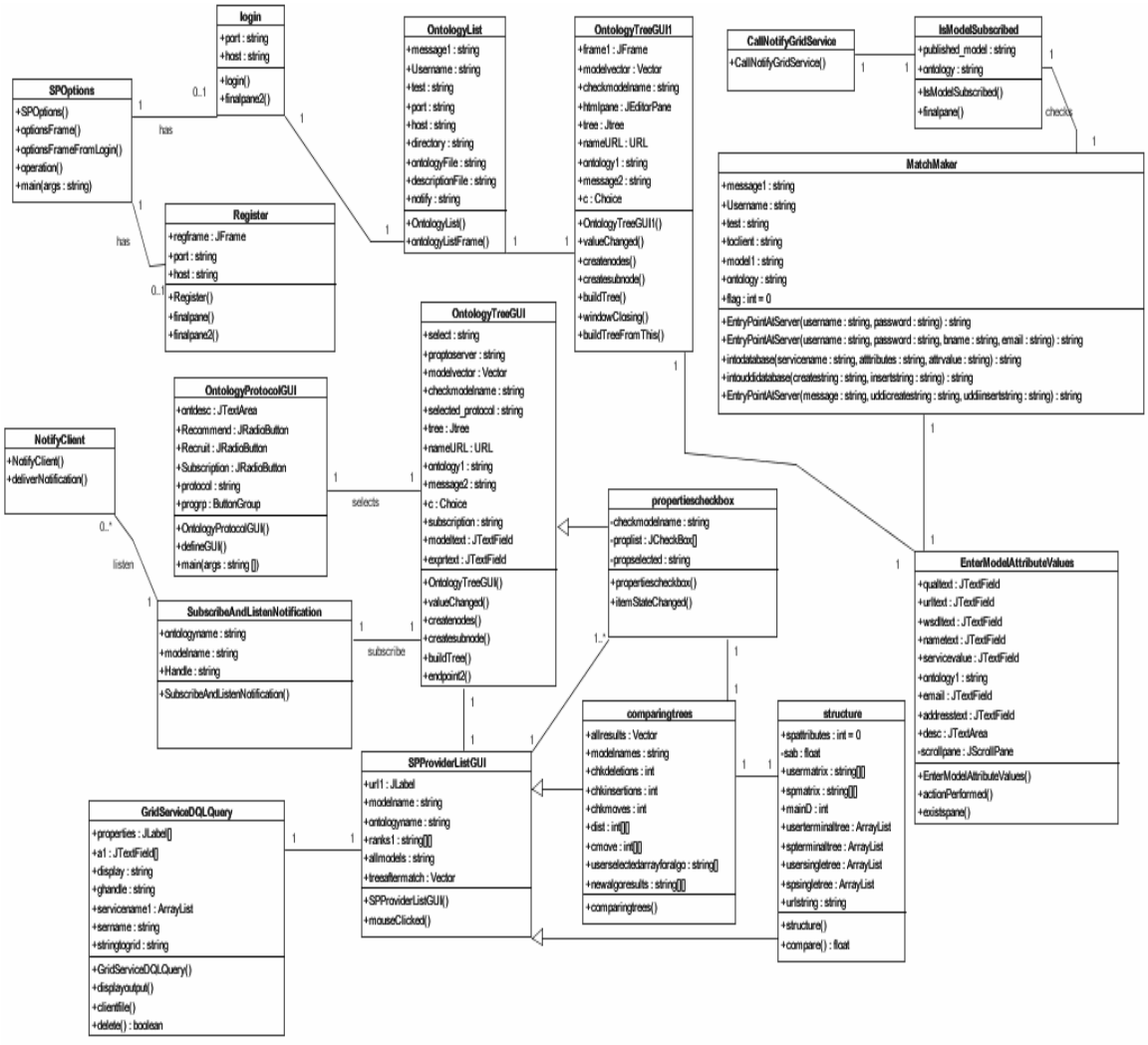

Figure A.1: Design Class Diagram of ABMS

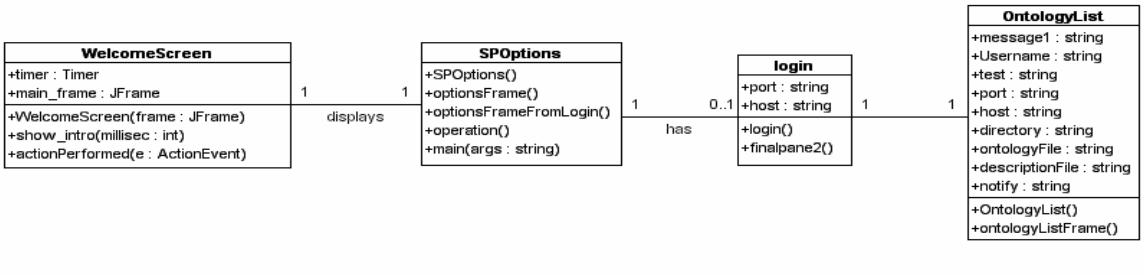

Figure A.2. Modeling "Login" phase.

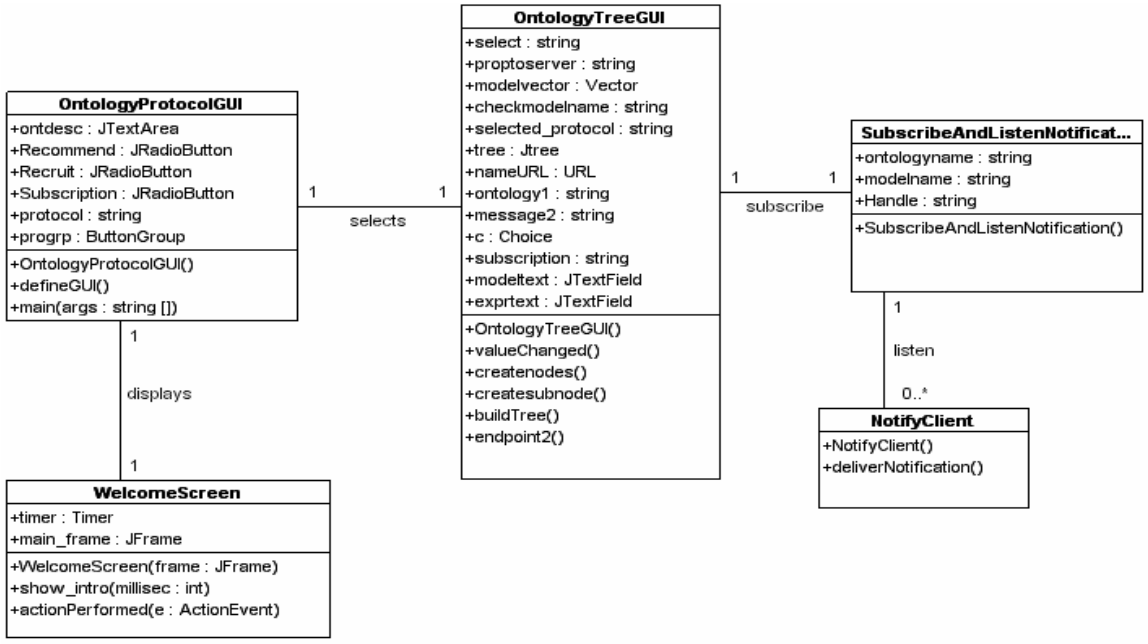

Figure A.3. Class Diagram modeling subscription to a model

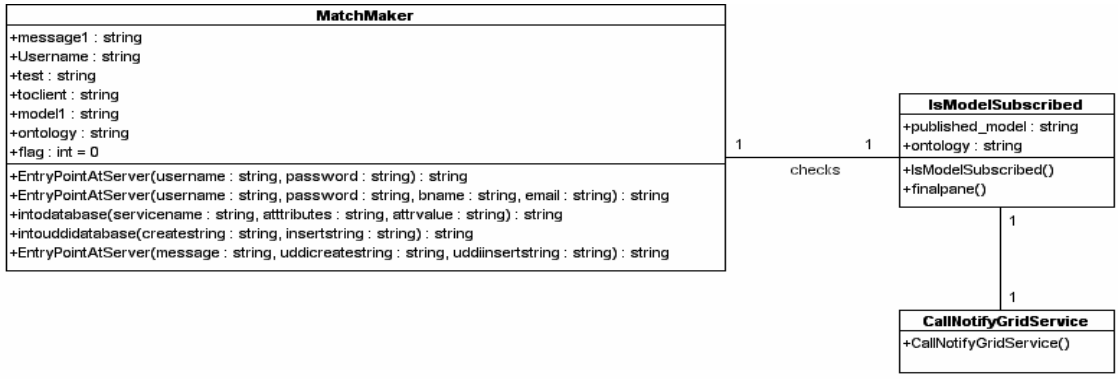

Figure A.4. Class Diagram: Check available subscriptions.

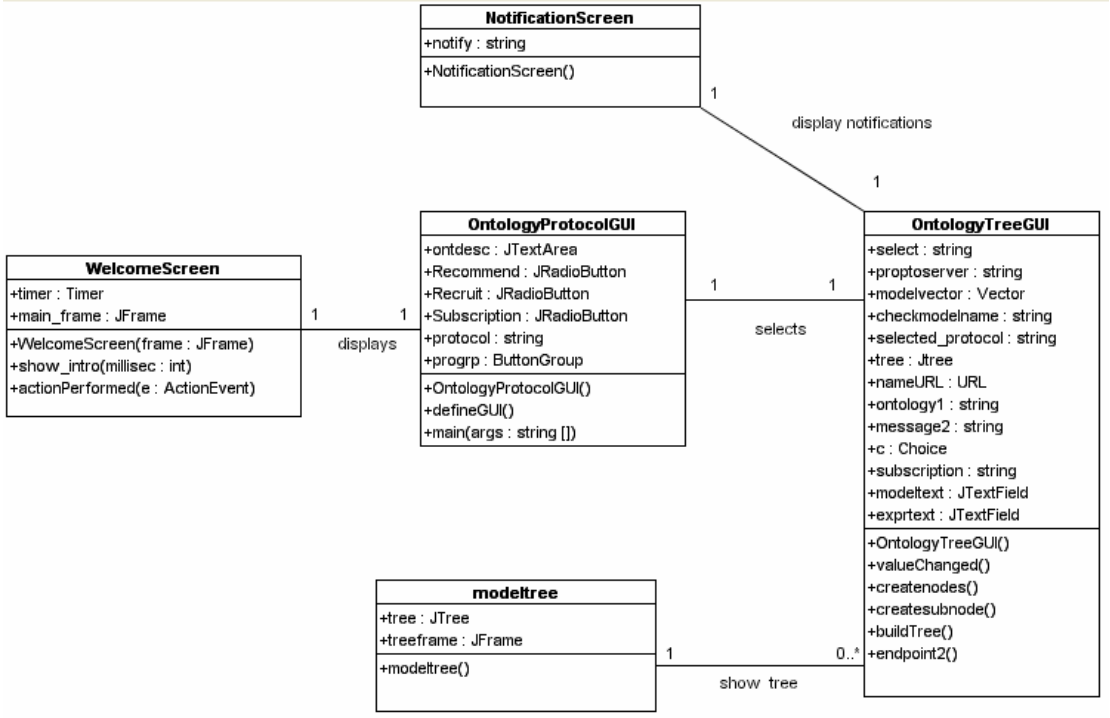

Figure A.5. Class diagram: Display available notifications

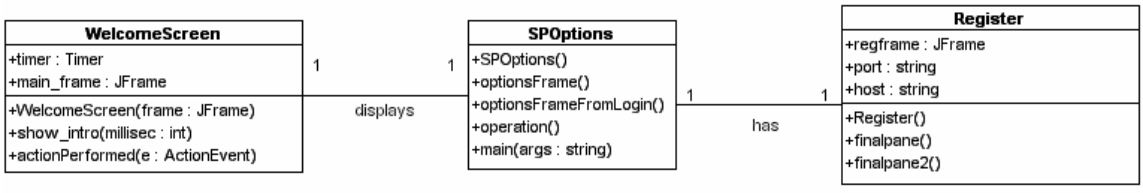

Figure A.6. Class Diagram: Registering a Service Provider

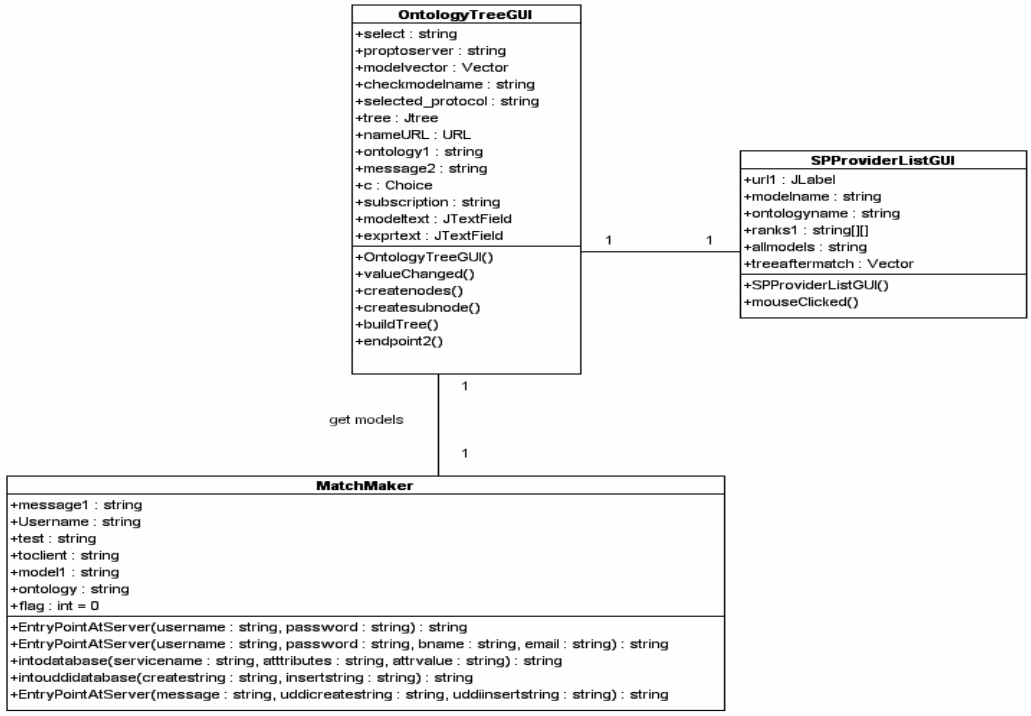

Figure A.7. Class Diagram modeling first level matchmaking