SERVICE-CENTRIC BROKERING IN DYNAMIC E-BUSINESS AGENT COMMUNITIES

Sumi Helal, Mei Wang and Arun Jagatheesan
Computer and Information Science and Engineering Department
University of Florida, Gainesville, FL 32611
Tel: 352-392-6845, Fax: 352-392-3238
{helal,mwang,aswaran}@cise.ufl.edu

Abstract

The paper focuses on the development of protocols for brokering-based agent communities in e-Business. The global economy is driving e-Business to become service-centric offering highly modular services that can be flexibly and dynamically composed of rapidly deployable e-Services. This trend is giving rise to a new set of requirements of negotiation-based, autonomous, and intelligent computing. Therefore, it is expected that in the near future, e-Services will be designed and implemented as software agents (also known as agent-based systems). This paper prepares for the proliferation of agent-based systems in e-Business by contributing a suite of protocols for self-organizing agent communities. The protocols are based on a three-tier architecture of agents, brokers, and superbrokers. We present the architecture and the protocols (*Broker-Based Agent Community Protocol*s, or BBACP). An implementation using *JKQML* is also presented along with a case study drawn from the electronic auto-trading domain.

I. INTRODUCTION

If we consider the use of money instead of barter system as the first major milestone in the history of business, E-business would be second one. The development of e-Business has been observed in several ways: flexibility requirement in the business market, emergence of virtual enterprises or net enterprises, and globalization of business [14][11]. If we forget the hype associated with this technology, and analyze the web-based e-Business applications and systems developed by individual companies, we find they are neither compatible nor interoperable with each other. New mechanisms and technologies need to be invented in order to make individual standalone e-Business interoperable and cooperative. The new mechanisms must support the extension of B2B exchanges into exchange-to-exchange (X2X) models. In our view, agent technology is one of the core technologies that will accommodate the growing proliferation of e-Business.

Agent technology has been widely adopted in the artificial intelligence and computer science communities. An agent is a computational system that operates autonomously; communicates asynchronously; and runs dynamically on different processes in different machines, which support the anonymous interoperation of agents. These qualities of agents make them useful for solving issues in information intensive e-Business, including speaking ontologies, advertising, service exchange, and knowledge discovery, etc., In a dynamic e-Business environment, the interoperation and coordination across distributed services is very important. The desire for more cost efficient and less sub-optimal business processes drive the use of agent technology in e-Business.

E-Business agents can provide e-Services as well as exchange information and e-services with other agents. The interoperation of e-Business agents leads to the formation of the e-Business Mall, which is an interaction space of agent communities under various business domains.

The significant problems in the e-Business are information deficiency and asymmetry between the business participants. In the formation of an virtual enterprise or a business community, each participant faces difficulties in exchange of information on products and services in an efficient manner. We

contribute the concept of agent community, in the context of e-Business as an approach to those problems. This approach befits the current necessity of forming marketplaces that can be scaled to exchange-to-exchange(X2X) models.

An e-Business agent community is a self-organized virtual space consisting of a large number of agents and their dynamic environment. Within a community, highly relevant agents group together offering special e-Services for a more effective, mutually beneficial, and more opportune e-Business. Each agent community consists of agents specializing in a single domain/sub-domain, or highly intersecting domains.

The ideal e-Business agent communities aim to

- make it easy to develop virtual enterprises, on which companies are joining and sharing resources and business processes for the production, marketing, and other services, provide protocols for the exchange of information, products and services among business entities
- represent certain potential relationships among potential members, and support the interoperation among them
- help in the establishment of relationships among agents with common interests
- provide discipline in the open network in anticipating of e-Business proliferation
- accommodate solutions to issues such as trust, efficiency and credentials

In section II, we present the layered architecture of agent communities and describe the specification, analysis, and design of agent organizational structure. Section III addresses the role of ontologies in agent communities. Section IV presents the components of the *Broker-Based Agent Community Protocol*s (BBACP) in detail. The protocols or rules in the service-centric *Brokering Layer* include processes of joining e-Business in the community, knowledge discovery and exchange, and advertisement of services or capabilities. In section V, we illustrate how the community and the supporting protocols are designed and implemented using the *JKQML* [6] agent shell. We present the functionality of the community components and our implementation of the layered community architecture and protocols as Java APIs supporting interoperation within an agent community. Finally, an e-Business case study based on the BBACP protocol is demonstrated, showing the benefits of service-centric brokering in self-organized agent communities.

II. Architecture of Agent Community

An e-Business community needs to have a loosely coupled architecture with heterogeneous components. The architecture must satisfy the demands of flexibility, scalability and interoperability in an e-Business environment. We introduce a hierarchical brokering architecture for the e-Business agent communities that will act as the intermediary between the supplier and the consumer of services. The community is organized as layers. An agent community (as shown in Fig.1) will consist of an *Agent Layer*, a *Brokering Layer* and a *Super Brokering Layer.*

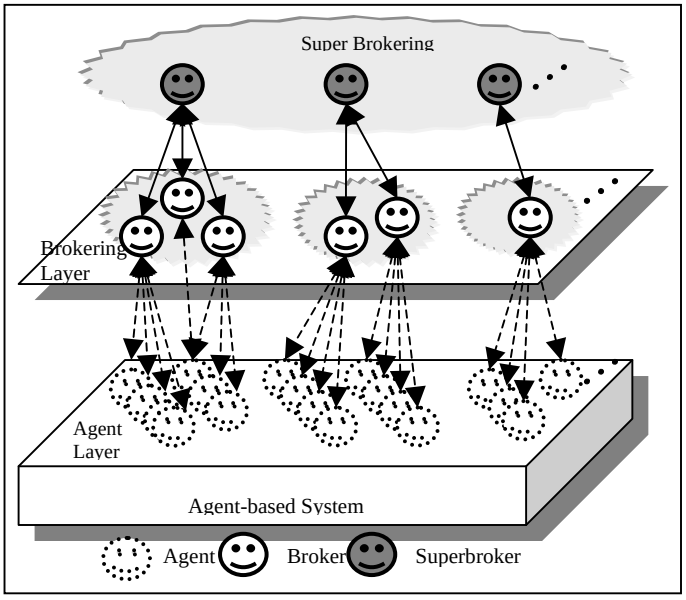


Figure 1. Hierarchical Architecture of an Agent Community

The *Agent Layer* is the bottom layer in the community architecture hierarchy. Each agent or agent based system (ABS) in the *Agent layer* interoperate with each other and they provide specific e-Service(s) that can be used by the community. These service-centric agents have information about a product or service under a certain knowledge domain. They self-organize themselves together with a broker from the next layer. The broker facilitates the business processes among agents. Thus, a broker can be viewed as the representative, for a set of agents in the *Agent Layer* with common interest in the specified domain.

The broker acts as the facilitator of information and services exchange among the heterogeneous agents. Agents communicate their needs and capabilities to the brokers. Brokers use the acquired knowledge about the other agents in the community, to reply or route messages to the appropriate agents. A brokered agent system provides a dynamic structure as agents can join or leave the agent community at any time. Also, a brokered agent system can provide a flexible coordination in a loosely coupled system of heterogeneous agents.

The interoperation of various brokers in the *Brokering Layer* forms a multi-brokering system, which consists of multiple brokers and a superbroker. Superbroker is a broker in the next layer supporting the knowledge sharing efforts among the registered brokers and taking care of the organization of the community. Every broker in the brokering layer registers with the superbroker of the community using a protocol (mentioned in the section IV). When the service requests can not be satisfied within a brokered agent system, the request is sent to other brokers within the community. The capable brokers in the community provide the requester with the information needed to obtain the requested service. This leads to the expansion of knowledge among the requesting brokers. Also, brokers can make subscriptions, which are requests to be notified when the right capabilities are available in the community. The superbroker maintains the service subscriptions and notifies the subscribers when the services become available. Individual brokers maintain their autonomy facilitating the interaction of their member agents. The *Brokering Protocol* governs how the brokers advertise capabilities and services of agents they represent; how they reason and select which broker is appropriate for the requested services; and how they communicate and coordinate over common ontologies of the community.

On the top of the community hierarchy is the *Super Brokering Layer*. In this layer, superbrokers interact with each other directly or through superbroker consortia, which eventually leads to the concept of

virtual enterprises or X2X model e-business. In this layer, superbrokers operate under the *Super Brokering Protocol* , which governs how superbrokers communicate and coordinate over common ontologies in the community. The protocol also governs how the superbrokers reason and solve the community issues, such as trading brokers or other community restructuring.

The community is easily scalable by accepting new brokers that represent a set of agents. When a new set of agents represented by a broker joins the community, the superbroker becomes the entry point for interaction to the new broker. When a broker joins or leaves the community, the change of the knowledge and services will be updated in an information repository maintained by the superbroker and known to the existing member brokers. Superbrokers have tight control over load balancing, support great service reliability, and provide potential wide ranges of services in the e-Business community.

Researchers have designed different organizational structures for brokering systems, such as the peer-to-peer multi-brokering architecture in *InfoSleuth* [10], the partitioned repository design in blackboard systems [2], centralized message routing design in *JAT Lite* [7], and the multi-facilitator mediation in *OAA* [9]. Comparatively, the hierarchical agent community is designed to meet the requirements of the proliferation of e-Business agents, to accept new brokers without the direct necessity of adjustment on the Agent Layer. In the Brokering Layer, brokers communicate with each other directly or indirectly under the superbroker supervision. The communication and interaction protocols specified in Broker-Based Agent Community Protocol are capable of avoiding a single point failure, unlike those in *JAT Lite* and blackboard systems. Each broker only needs to have information about the superbroker it is registered with, and communicates with other brokers only upon necessity. Thus the amount of messages flowing among the brokers within the community is minimized, providing the faster solution to the network bandwidth restriction imposed on various e-Business or mobile business applications.

III. The Role of Ontology

In order to self-organize into communities, each broker needs to know the spoken ontologies of the community. Hence, every broker should contain a generic broker instance, which implements at least the basic infrastructure needed for an agent community. Some assumptions are necessary to the joining protocol for the brokers:

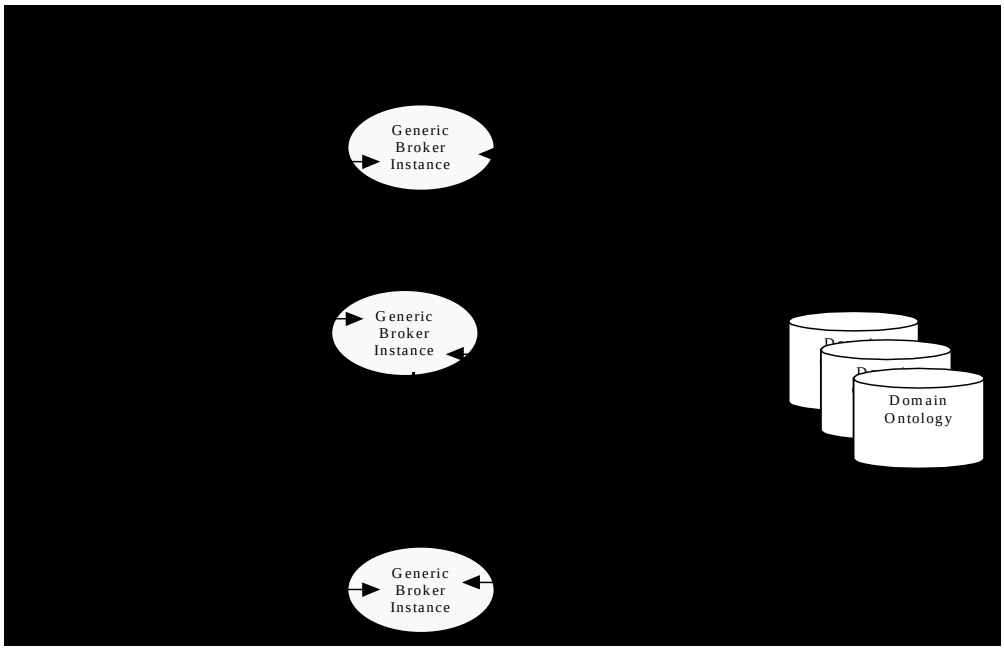


Figure 2. Universal Community Ontology, Domain Ontology and Community Specific Ontology

Each generic broker instance, which has a *Universal Community Ontology*, can access various publicly available *Domain Ontologies* in a community. *The Universal Community Ontology* is the fundamental ontology to be used by all e-Business communities, and provides brokers the minimum amount of vocabularies and knowledge towards self-organizing within a community. The *Universal Community Ontology* enables brokers to interact with a superbroker, conveying the spoken ontologies and knowledge capabilities among other information. *Domain Ontology* provides vocabularies and knowledge within an e-Business sub-domain.

In each e-Business community, the *Community Specific Ontology* should be made available and open to both community members and applicants. Thus, prospective community members are able to browse and learn about the available *Community Specific Ontology* maintained by the superbroker before initiating any application. Based on *Community Specific Ontology*, the applicant broker can determine the relevancy between its own knowledge and that of a community, as well as other self-organizing factors. Only if knowledge relevancy and value added are expected, would further procedures need to proceed.

As shown in Figure 2, the *Universal Community Ontology* and *Domain Ontologies* that are possessed by the applicant broker enable it to initiate an application procedure. Applicant brokers have the goals of learning more about the *Community Specific Ontology* and other community facts. The superbroker maintains the *Community Specific Ontology* and acknowledges them to the applicants during the joining procedure. The joining procedure occurs when an applicant broker sends a membership application to the superbroker. In a service-centric agent community, membership application presents its knowledge as a constrained sub-domain of the e-Business domain. By using the *Universal Community Ontology* and *Domain Ontology*, the broker provides the general information about itself(e.g. name, location, status) and capabilities (e.g. knowledge of a sub-domain, e-Business transaction types, service credential characteristics). Currently, the *Universal Community Ontology* is implemented and described using KQML [4] as low-level language. These Ontologies help in the organization of the communities and the protocols for the communities.

IV. PROTOCOLS FOR AGENT COMMUNITIES

We introduce the *Broker-Based Agent Community Protocol*s (BBACP) to enable inter-organizational business processes. The BBACP consists of an *Agent Protocol*, a *Brokering Protocol*, and *a Super Brokering Protocol*, which provide mappings from states to action in each agent community layer.

The *Agent Protocol* involves interaction with other agents, which varies so as to suit to the business application and usage scenarios. Each agent-based system follows a local protocol, the *Agent Protocol*. A local protocol is a function from local states to local actions, such as applying rules or sending messages [11]. Therefore the *Agent Protocol* is the base-level commitment and responsibility to the community. Because it is an open protocol, much space is left for negotiation when agents are built. The autonomy of the agents themselves also requires self-government and self-organization. By accepting the *Agent Protocol*, agents commit themselves to play special roles and to guarantee properties such as sincerity and producing appropriate responses [5].

The *Brokering Protocol* extends the brokering function to meet the needs of interoperability among the brokers and with the superbroker. The *Brokering Protocol* describes a cooperative multi-brokering system, which provides the solution to interoperation among brokers in a dynamic heterogeneous agent community. Each broker performs basic brokering functionality (such as service discovery and knowledge sharing) within a set of e-Business agents. The broker also represents e-Services of the set agents for which it acts as an intermediary in the community. Individual brokers representing a set of agents are allowed to advertise their business service and send capability queries to other brokers, as well as receive advertisements from other brokers. In addition to regulation of joining in and leaving from the community, other issues covered in the Brokering Protocol include business knowledge discovery, sharing of acquired knowledge, and information load control.

The *Super Brokering Protocol* governs how a superbroker communicates and coordinates in a virtual enterprise; how they reason and solve the community issues; and how to form and maintain the superbroker consortium.

Our current research focuses on the *Brokering Protocol* layer, which includes the following protocol components.

IV-A. The Joining Protocol

The protocol for a broker joining a community includes a logic conversation between the broker and the superbroker. The conversation policy (or rule) is a sequence of rule based communication which specifies the social behavior of the applicant broker and the superbroker at each state. The conversation policy is mapped into a *Finite State Machine* (FSM) as illustrated in Figure 3.

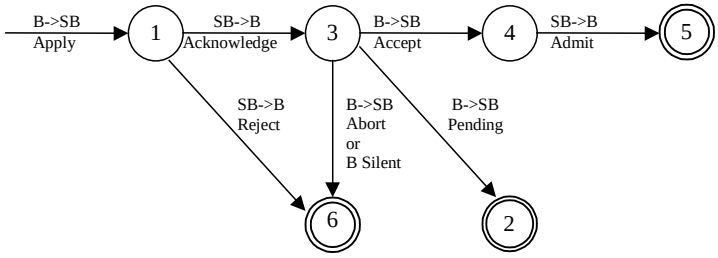


Figure 3. FSM for Community Joining Protocol

Each broker, in its application to the community, declares to the community what kind of ontology concepts it can deal. After receiving a registration attempt (State 1), the superbroker conducts multi-faceted membership evaluation of the applicant such as creditability, possibility of value-added to the community, potential expansion, knowledge and capability relevancy, among others. The conclusion of the evaluation is either reject (State 1 -> State 6) or acknowledge (State 1 -> State 3). The acknowledgement message contains three components: *Domain Ontology* and *Community Specific Ontology*.

At State 3, the broker receives the acknowledgement from the super broker and automates its behavior related to ontology adjustment or other knowledge changes. The broker can now makes a final conclusion to join the community or not. If a positive conclusion can be made immediately, a message notifying the acceptance of *Domain Ontology*, *Community Specific Ontology* is sent to the superbroker (State 3 -> State 4). The finite state (State 5) is reached when the superbroker confirms the grant of membership upon receiving the acceptance message. If a negative conclusion is made, an abort message is sent to the superbroker (State 3 -> State 6), which indicates the end of the conversation resulting in the finite state (State 6) being reached.

If a decision can not be made by the broker whether to join the community or not, a pending message is sent to the superbroker. In order to allow the applicant to continue the member application in the future, the superbroker must maintain the knowledge of access history. It is clear that the state information should be kept and used in guiding conversation policy. The question is where the state information should be kept. Our approach is to embed state information in the conversation policy in the superbroker side. During the application processing in the future, the superbroker will check the applicant access history, thus following the conversation policy based on state information. If the superbroker does not receive any replies within a specified period of time, the synchronous conversation will move to the finite state of application failure (State 6).

Upon approval of a successful registration, the superbroker updates the new broker in its repository. Failure of registration could encourage broker to engage in additional learning activities. Thus the superbroker manages the relationship between ontologies and brokers, maintaining consistency in ontology among members.

IV-B. The Departure Protocol

The departure protocol is used to inform the superbroker that the broker won't be able to provide the advertised service(s) anymore. The leaving broker initiates the leaving procedure by sending an "UNADVERTISE" message to the superbroker. The knowledge and services available to the community are updated by sending "TELL" messages to all members. Alternatively, the leaving broker can broadcast the "UNADVERTISE" message to member brokers directly. Upon receiving the "Unadvertise" message, the superbroker and the other brokers remove the services advertised by the leaving broker from their repositories. The knowledge the leaving broker acquired from the community is also removed.

The leaving broker finally sends an "UNREGISTER" message to the superbroker. The superbroker removes the broker's entry from the member repository and updates its member credit system. The member credit system is the repository maintaining the history and service quality of previous and existing members. It acts as a quality monitor to ensure the knowledge sharing efforts in the community and prevent the malfunctions of community members.

IV-C. The Knowledge Discovery and Exchange Protocol

When a broker can not satisfy any service request within its brokered agent system, the request is sent to other brokers within the community. The capable brokers in the community provide the requester with information about the service. This leads to the discovery and expansion of knowledge among the requesting brokers. The knowledge discovery is initiated by the inquiring broker, who multicasts the service request(s) to the community. Upon receiving the request(s), the capable brokers reply with appropriate service information that enables the requester to communicate with the service providers and obtain services from them directly. In our implementation, the inquiring broker sends a request to a superbroker. The superbroker searches its capability repository for service(s) that satisfy the request. If there exits at least one match, it sends the result as a "REPLY" KQML message to the requester. The received service information contains the required parameters to communicate with the service provider through KQML messages. Thus the requester is able to directly negotiate with the service providing broker(s) using specified language and ontology. The amount of knowledge discovered is based on the brokers' objectives: ASK-ONE (one capable service provider is recommended), ASK-ALL (a list of service providers are returned).

A broker can also request to be informed if some specific service becomes available in the community, which is another approach of knowledge discovery. This form of knowledge discovery called subscription is presented in the performative "SUBSCRIBE". A broker may broadcast a subscription to the community, so that each member broker stores it. If a broker is capable of serving the subscribed service, it will reply to the subscription and start a conversation with the message sender. Otherwise, the receiver does not respond and stores the subscribed subjects in its subscription repository. When a broker, or an agent it represents, develops new capabilities that meet the specification of the subscription, the broker will notify the subscriber and initiate conversation. In our implementation, the superbroker acts as a special broker that maintains all the capability information in the community. When the superbroker receives a subscription, it searches the community capabilities repository to see if any of the services available in the community matches the request(s). It also checks if the capable broker is up and running. Then, the requested information is sent to the subscriber by using the "TELL" performative. If no knowledge is able to serve the request, the subscription is stored in the superbroker's subscription repository. When a new service satisfying the subscription is advertised in the community, the superbroker notifies the subscriber(s) of the availability of the new capabilities. With this received information, the subscriber can then obtain the service by initiating a conversation directly with the service provider(s). Unsubscription is used to inform the community that a broker no longer needs to be informed for the subscribed service. A broker can unsubscribe one service at a time using the "UNSUBSCRIBE" performative. The "UNSUBSCRIBE" message is either multicast to all community members directly or sent to a superbroker and consequently sent to member brokers. The receiving brokers remove the subscription from their subscription repositories in both the approaches. We implement the latter approach. When the superbroker receives an unsubscription message, it removes the corresponding broker's entry from the subscription repository so that no more notifications are sent to that broker.

Knowledge discovery follows knowledge discovery rules (a finite state machine). If a request is satisfied and a discovery is made, the broker updates its acquired capabilities repository, so that it can use the acquired knowledge later; if a request is unsatisfied, the broker receives a "SORRY" message. it could refine the request content and re-send it to the superbroker.

IV-D. The Capability Advertisement Protocol

A broker can advertise its capabilities to the community by first sending the advertisement to the superbroker. The advertisement specifies the required information, which the broker interested in the advertisement has to use to get the desired service. The performative used is "ADVERTISE". If a broker wants to advertise multiple services, it does so by sending multiple advertisements. The advertisement is broadcast to the community by sending an "ADVERTISE" message other member brokers.

When a broker receives an advertisement, it updates its repository by adding a new entry, inserting the name of the broker, the credential information, and domain dependent information. If the advertised service is new, or an update is made to an existing advertisement, a "TELL" message is sent back to acknowledge the advertiser. A "SORRY" message is sent to inform the advertising broker about any failure is adding the advertisement.

The unadvertisement process is triggered when changes are made to any advertised services provided by an agent. When a broker receives a service change message from its member agent, it updates its self-capability repository, which means either removing the entry permanently or updating the characteristics of the particular service. The broker then sends an "UNADVERTISE" message to the community to indicate the necessary changes needed to keep member's information repository up-to-date. In our implementation, a superbroker has much broader knowledge and maintains all the advertisements in the community. Services are advertised and unadvertised to the superbroker and also to the interested members. Since the superbroker maintains the subscriptions from its members, only the subscribers need to be sent messages instead of broadcasting to all community members. When a superbroker receives an advertisement, it first updates the entries in the capability repository. Then it checks the subscription repository to see if the advertised service satisfies any subscriptions. If one or more matches are found, the superbroker notifies the corresponding broker(s) by sending a "REPLY" message in, which specifies the service information requested. When a member agent needs to make a change in service it had previously advertised, it sends a message to its broker, which sends an "UNADVERTISE" message to the superbroker to indicate the necessary changes needed to keep the information repository up-to-date. The superbroker updates the community capability repository by either removing the entry permanently or by updating the characteristics of this particular service. The superbroker also sends notifications to member brokers who have acquired or subscribed to that particular service.

V. IMPLEMENTATION

In our implementation, agents operate and interact in the infrastructure provided by the agent shell—*JKQML*. Figure 4 shows the design and implementation of our agent community architecture.

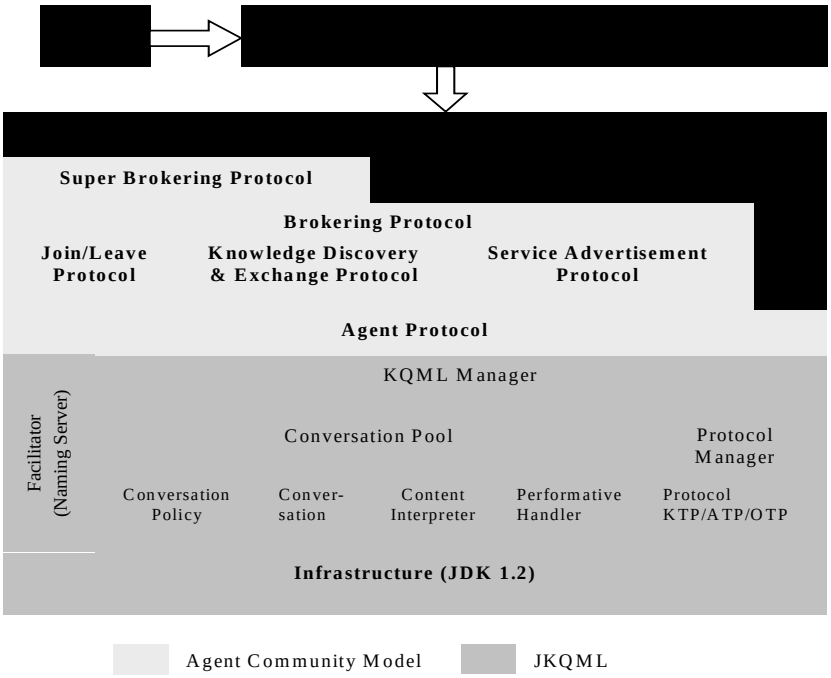


Figure 4. Agent Community Architecture Design and Implementation

The lowest layer is the *JDK 1.2* that can execute Java bytecode in any open network through Java Virtual Machine. Above the Java infrastructure rests the Java-based KQML (*JKQML*) [6], the agent shell (or agent development toolkit). *JKQML*, developed by IBM is a framework to construct KQML-speaking software agents, to build loosely coupled distributed systems by using an agent communication language—KQML [4], to provide flexible interoperability for exchange of information and services between software systems. *JKQML* is written entirely in the Java language to construct software agents communicating over the Internet. It is designed to support various transport protocols, and to provide flexibility for the extension of the framework. As shown in the Figure.4, *JKQML* provides interfaces to process KQML messages for agent application through the *KQML Manager*. The agent applications can access other major components such as conversation pool and protocol manager through the *KQML Manager*'s interfaces. Currently *JKQML* supports three transport protocols: the *KQML Transfer Protocol* (KTP) for the ordinary TCP/IP environment; the *Agent Transfer Protocol* (ATP) for the *Aglets* environment [1]; and the *Object Transfer Protocol* (OTP) for transferring Java Object.

The agent community model defines how agents interact with each other in a cooperative manner. It utilizes the *KQML Manager* and extends the functionality to support the BBACP, which is composed of three sub layers: *Agent Protocol*, *Brokering Protocol*, and *Super Brokering Protocol*. Each protocol governs one of the layers of the agent community and its interaction with the other layer. The *Agent Protocol* is the base-level commitment and responsibility to the community, which differs among different business providers. Because it is an open protocol, much space is left for adjustment when agents are built. The middle layer of the community model is the *Brokering Protocol* that provides the inter-brokering among community members. The upper level of the community model is the *Super Brokering Protocol*, which extends the concept of agent community of virtual enterprise to the e-Business Mall.

In our implementation, the Java objects like *CRAgent*, *Broker*, *SuperBroker* and *SocialViewer* captures one or multiple functions in the community. There are two inheritance relationships in Figure 5. The *Broker* class inherits the *Agent* class and the *SuperBroker* inherits all the functions of the *Broker* class. Inter-agent and inter-broker interaction is also illustrated.

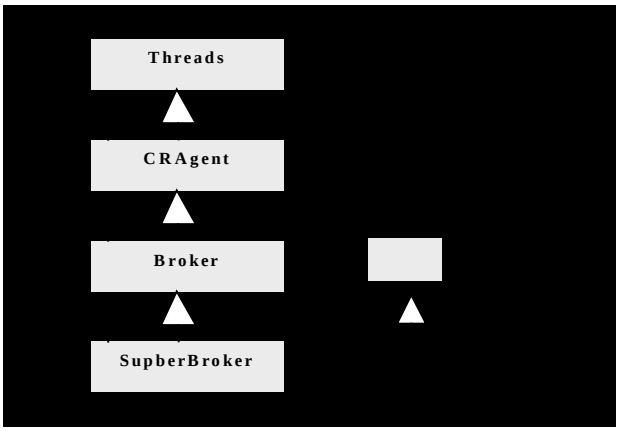


Figure 5. Community Components Diagram

Each broker object maintains three information repositories: self-capabilities repository, acquired-capabilities repository, and subscription repository. The superbroker, maintains the service information repository, guarantees relevance and affinity, enforces interoperations and knowledge sharing efforts, and tracks member brokers' credential characteristics. The credential information repository contains the broker name, location, ontologies, status, cost of services, reliability, and quality. By maintaining credential information of member brokers, the superbroker can initialize a membership evaluation procedure. All members vote on the quality of every membership and submit voting results to the superbroker. Based on the voting result, the superbroker expunges the membership of unqualified broker(s). This makes sure that only qualified brokers are present in the community.

The social viewer is a special agent that provides a domain-independent social view service. It is not an essential part of our architecture.It views and analyzes social roles of agents, brokers, and superbrokers. It shows their organizational inter-relationship, and logs the message exchange and collective behaviors between brokers and superbrokers during their interoperations.

Our current research focuses on the middle layer of community protocols, the Brokering Protocol layer. Our implementation provides community-ready interface, and supports the self-organizing feature in open e-Business communities. The community-eady interfaces are sets of APIs, which allow software programmers to easily write the agent applications that inherit the functionality of the organizational roles needed in the agent communities. The complexities and details of the protocol implementation are hidden from programmers. The APIs allow programmers to extend the classes and to overwrite the methods so as to implement more functions for specific business domains while maintaining the essence of brokered community protocols.

Appendix A summarizes the community-ready API for the BBACP protocols. It shows the API components of each layer and describes component interactions through KQML performatives. More detailed explanation of the API implementation can be found [13].

VI. CASE STUDY

We demonstrate an application of the agent community and the community protocols in the on-line auto-trading domain. The objective is to enable the business participants to electronically buy and sell cars over an open network via e-Business agents.

To present the characteristics of the business process of buying and selling cars, we assume there are several brokers, one superbroker and a variable number of agents. The size and type of brokers are not limited to the description. As an open network, the auto trading community allows new brokers to join based on their knowledge relevancy and value-added to the community. Currently we have designed a simple community consisting of participants such as: Individual Buying Broker, Dealing Broker, Individual

Selling Broker, Manufacturer Broker, Warranty Broker, Auto Transport Broker, Vehicle History Broker, Auto SuperBroker, White pages and Yellow pages.

As described in the domain specification, all brokers automate the trading process among themselves. However, the direct business protocols are facilitated and maintained by the superbroker of the community, the AutoSuperBroker. The multi-brokering trading community contains several roles. It is possible for each individual role to be played by an individual broker. Also, a single broker can play several roles. Each role played by a broker contains some responsibilities, such as sending registration in order to join the trading community, advertising its services or requesting one, and interacting with other type of brokers. This section introduces the role responsibilities for each broker including Member Applicant, Membership Inquirer, Services Advertiser, Service Inquirer and Subscriber and Broker. Tables 1 to 5 describe each responsibility in detail.

Table 1: Auto Trading Community Role Descriptions—Membership Applicant

ROLE	MEMBERSHIP APPLICANT	
Role Model	Auto Trading Community	
Description	Register with AutoSuperBroker and obtain community ontology and bylaws	
Responsibilities		Collaborators
	To send membership application message To send leaving community message	Destination Role: AutoSuperbroker
Explanations	By default , each broker is responsible for registering as a member in order to participant the any trading	

Table 2: Auto Trading Community Role Descriptions— Membership Inquirer

ROLE	MEMBERSHIP INQUIRER	
Role Model	Auto Trading Community	
Description	Query with AutoSuperBroker	
Responsibilities		Collaborators
	To query and receive member information of a named broker or a list of brokers	Destination Role: AutoSuperbroker
Explanations	By default , each broker is capable of querying and notifying	

Table 3: Auto Trading Community Role Descriptions— Services Advertiser

ROLE	SERVICES ADVERTISER	
Role Model	Auto Trading Community	
Description	Advertise services the broker represents to AutoSuperbroker or direct to Brokers	
Responsibilities		Collaborators
	To send advertisements or unadvertisement message of agent capabilities and services it represents	Destination Role: AutoSuperbroker Broker
Explanations	Each broker advertises the services it represents through AutoSuperBroker or direct to multiple brokers. The business relationship usually groups a set of brokers as service partner. Brokering protocol specifies how the service advertisement are represented.	

Table 4: Auto Trading Community Role Descriptions—Services Inquirer and Subscriber

ROLE	SERVICES INQUIRER AND SUBSCRIBER	
Role Model	Auto Trading Community	
Description	Request the AutoSuperbroker or other brokers for services that could not be handled by broker itself.	
Responsibilities		Collaborators
	To query and obtain brokers' information with certain service from the AutoSuperBroker, To query and negotiate contract specifications with particular broker(s), To subscribe to and obtain certain types of services	Destination Role : AutoSuperbroker or Broker
Explanations	Each broker requests services information from AutoSuperBroker and conducts business contracting with one or multiple brokers. Each broker provides the services and respond any inquirer and subscriptions.	

Table 5: Auto Trading Community Role Descriptions— Broker

ROLE	BROKER
Role Model	Auto Trading Community
Description	Provide service brokering in particular brokered agent system, include process member applications and unregistrations, maintain service advertises of its members and provide capabilities matching and routing.

	The major goal is to hide all the multi-brokering interactions from member agents.	
Responsibilities		Collaborators
	To maintain directory of member and their abilities, To receive and process service advertisements and queries, To serve the request of agents without their direct involving the contracting. The interaction details among multiple brokers are hidden from the answers to agents	Source Role: Agent
	To negotiate service contracts with other brokers.	Source Role: Broker

The basic objective of the auto trading community is to model the business process chain , which adds value to all participants, and to support pair-wise contracts among multiple brokers to fulfill transactions. Figure 6 illustrates the relationships between organizational units, processes in the trading community. It also illustrates the information flow between these entities.

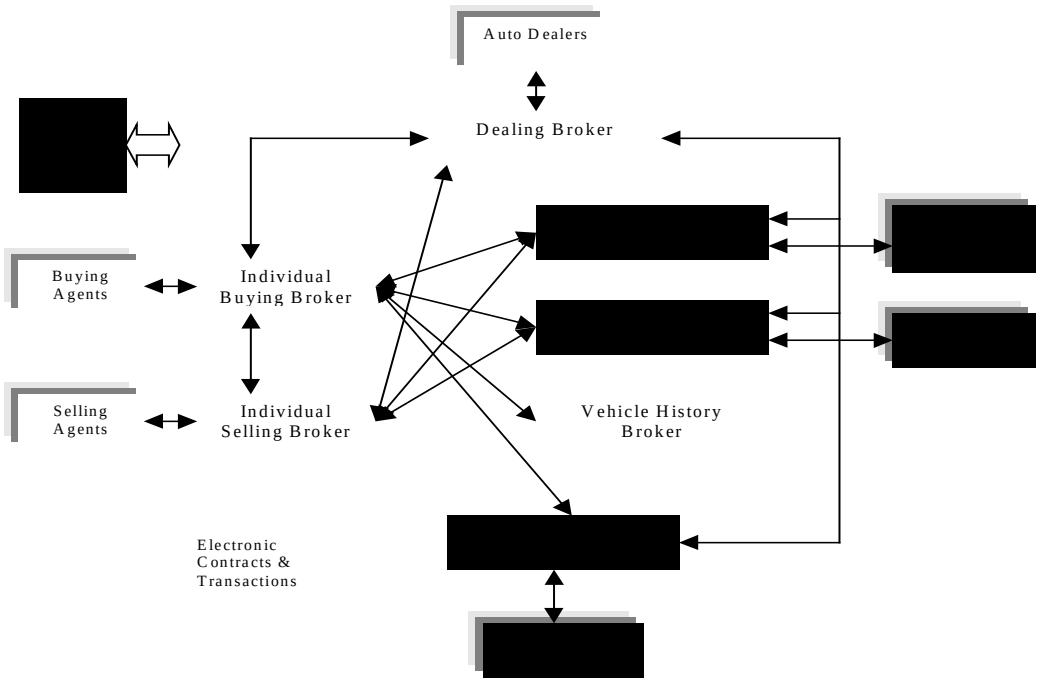


Figure 6. Interaction, Electronic Contract and Transaction in Auto Trading Community

Individual agents which are members of any brokered system, can take advantage of all the capabilities and services in the whole community. However, individual agents do not have to understand all the business process and contract specifications. All business deals can be made through the representative brokers. A buying agent registers with the Individual Buying Broker without any further knowledge about dealers, manufacturers, moving companies, etc. The Individual Buying Broker can buy from the Dealing Broker, the Manufacturer Broker, or the Individual Selling Broker, each of whom represents several specialized agents. In the best interest of their member agents, brokers interact and negotiate with each other based on the service knowledge they share using the BBACP protocols discussed. A service contract usually involves specifications of multiparty. For instance, the Individual Buying Broker buys a new Honda Civic from the dealer in Miami, while it makes a deal with the auto moving companies to transport the car from Miami to Gainesville at particular date with insurance. The car is actually routed directly from the

factory. Thus the dealer handles the additional shipping expenses. The car comes with a manufacturer warranty and six-year extended warranty with discount price from a warranty provider, the business partner of the car dealer. This particular buying process involves various brokers and dynamic service contracts among them. The layered protocols and the hierarchy architecture of the community capture the requirements of the business and helps in the organization of the auto brokering community. The members can use supply chain management and other logistic methods to further benefit from the community.

VII. CONCLUSION

A self-organizing community requires basic ontologies and protocols to support communications and interoperations among heterogeneous agents. The social nature of knowledge sharing carries high complexity. The capability advertisement and knowledge discovery should be achieved by message interaction among dynamic processes.

This paper proposes a hierarchically layered agent community architecture and a set of protocols guiding interoperations across the open network for e-Business communities. Also, a set of Java API for usage of these protocols has been provided [13].

We have contributed four community protocols that guide the conversations and interactions among different role players in the community. The knowledge discovery protocol and capability advertising protocol are used by serveice-centric brokers to enhance their capability in terms of their knowledge and ability to find services and solve problems within a set of agents. The superbroker and the applicant brokers use the joining protocol and leaving protocol to evaluate knowledge relevance and resolve the self-organizing issues in a dynamic e-Business environment.

The practical issues in setting up e-marketplaces are solved using e-communities by agents. The corporate boundaries do not stand as an obstacle in business integration, as we use virtual communities that need not connect directly to corporate data. Agents which have knowledge of the business partners are used to represent the processes and data of its corporate identity. Companies, instead of completely changing their existing systems, need to just send agents to the community and update their business processes based on the knowledge gained about suppliers or consumers of services. Since, the community can have any agent who can understand the ontology, the problems with anti-trust business practices in a global economy are also avoided. Thus, the agent communities are the answer to the current problems in establishing business to business marketplaces.

Knowledge or service information that can be gained is used as the basis of self-organization of the brokers in the community. Currently, we use simple methods to determine such relevancy or information gained. More sophisticated algorithms need to be developed to more accurately predict a broker's relevancy to the community. The community can use XML-schema for representing the domain ontologies of the community. Also, the usage of supply-chain management and other logistic methods to take full advantage of a e-community needs to be investigated to improve the business processes in the community.

VIII. REFERENCES

[1] Aglets, http://www.trl.ibm.co.jp/aglets/, Sept 1999

[2] Carver, N. & Lesser, V., "The Evolution of Blackboard Control Architectures," *CMPSCI Technical Report 92-71*, Oct 1992.

[3] Dan, A. et. al., "The Coyote Project: Framework for Multi-party E-Commerce," *Lecture Notes in Computer Science*, Vol.1513, pp873, Springer-Verlag, Heidelberg 1998.

[4] Finin, T., Labrou, Y. & Mayfield, J., "KQML as an agent communication anguage," In J.M. Bradshaw, editor, *Software Agents*, AAAI Press, 1997.

[5] Huhns, M.N. & Singh, M.P., "Readings in Agents," Morgan Kaufmann Publishers, San Francisco, CA, 1998.

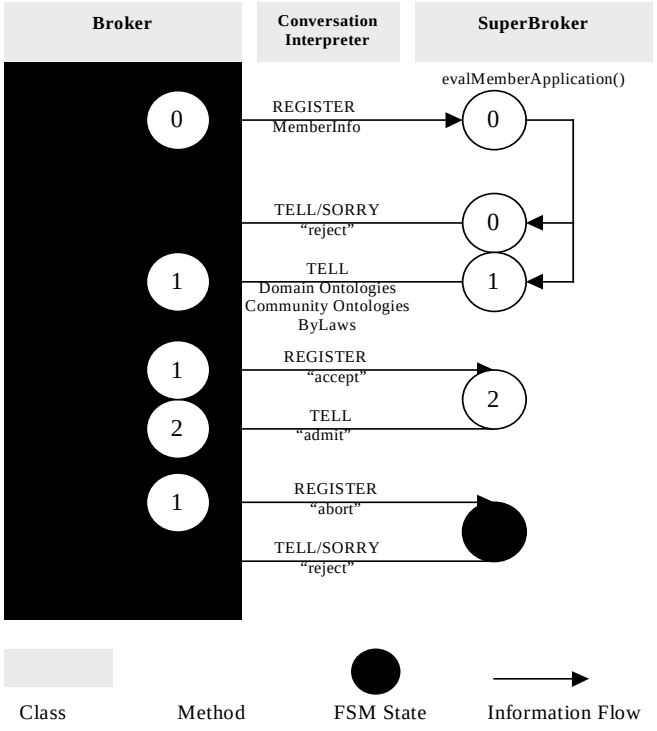
[6] JKQML, http://www.alphaworks.ibm.com/formula/jkqml, May 1999.

[7] JATLite, http://java.stanford.edu/, May 1999.

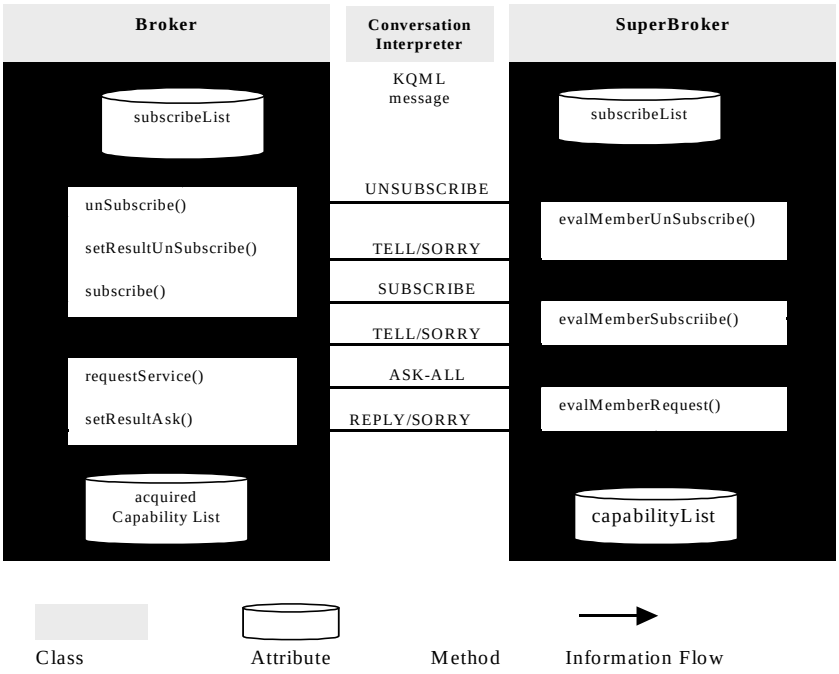
[8] Kendall, E.A., "Agent Roles and Role Models," http://www.labs.bt.com/projects/ibsr/papers/patterns/iaipm.ps.gz, Aug 1999
[9] Martin, G.L., Unruh, A., & Urban, S.D., " An Agent Infrastructure for Knowledge Discovery and Event Detection," Oct 1999.
[10] Nodine, M.H., Bohrer, W., & Ngu, A., "Semantic Brokering over Dynamic Heterogeneous Data Source in InfoSleuth," *ICDE 1999*: 358-365, 1999.
[11] Ouzounis, V., and GMD-Fokus, "R&D for New Methods of Work and Electronic Commerce," *State-of-the Art and Visions Workshops, Dec 1997-Apr 1998*, Brussels, July 1998, http://www.ispo.cec.be/ecommerce, Oct 1999.
[12] Vreeswijk, G.A.W., "Open Protocol in Multi-Agent Systems," *Technical Report CS 95-*, University of Limburg, Jan 1995.
[13] Wang, M., "Service-Centric Brokering in Dynamic e-Business Agent Communities," Master Thesis, CISE Department, University of Florida, Dec 1999.
[14] Zimmermann, H. D., "Business Media: A new approach to overcome current problems of Electronic Commerce," *Benbasat, Izak; Hoadley, Ellen: Proceedings of the 1998 Americas Conference on Information Systems AIS '98*. Baltimore, Maryland, Aug 1998.

IX. APPENDIX A

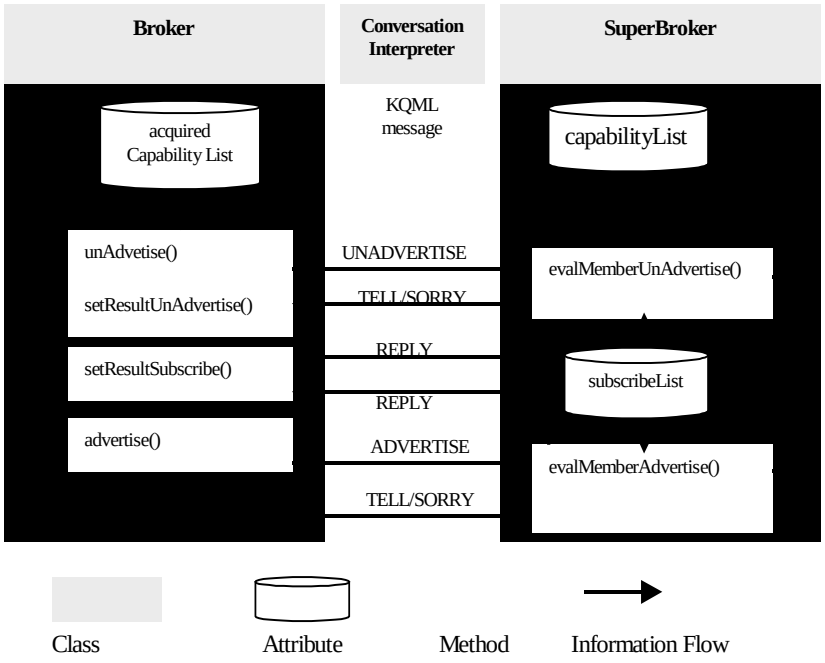
A-1. Summary of the Joining Protocol API



A-2. Summary of the Knowledge Discovery Protocol API



A-3. Summary of the Capability Advertisement Protocol API



A-4. Summary of the Departure Protocol API