Brokering Based Self Organizing E-Service Communities

Sumi Helal, Mei Wang, Arun Jagatheesan and Raja Krithivasan

Computer and Information Science and Engineering Department
University of Florida, Gainesville, FL 32611

{helal, mwang, aswaran, rkrithiv}@cise.ufl.edu
http://www.harris.cise.ufl.edu/projects.htm

Abstract

The rapid evolution of the Internet and its business tools is enabling the transformation and deployment of business processes as highly modular e-services that can be flexibly and dynamically composed to form ad-hoc workflow. This research prepares for the proliferation of automated, Internet-based workflow, by contributing a suite of protocols for self-organizing brokering communities that enables the discovery of relevant e-services. In this paper, we present protocols and architecture for e-service brokering communities, and discuss their use in workbase, an Internet-based, automated workflow system over e-services. The brokering protocols are based on a three-tier architecture of agents, brokers and superbrokers. We also present an infrastructure for dynamically composing new services from exiting e-services on the Internet. An implementation using JKQML of brokering communities is provided along with the architecture and design of our e-services and workflow concepts.

1. Introduction

Tools created during the Stone Age and the industrial revolution brought about lots of changes to the society. The quest for discovery of new forms of tools that can provide more services still continues. In today's world, an evolution of technologies is trying to bring electronic services (e-services) as tools, that can be accessed anywhere by any customer, business, agent, or device. In the near future, many of the day-to-day activities and e-business processes will be offered as individual e-services and will act as the virtual tools for our daily life. These e-services will be combined and reused, to create new, value-added services. In such a case, the new service can be viewed as the result of execution of a workflow of existing and/or newly created services. Our research goal is to enable the rapid and ad-hoc creation of automated, Internet-wide workflow using these e-services.

To achieve our goals, we need have a workflow engine, a process by which to discover available e-services, and an infrastructure to compose workflow in terms of e-services, and to invoke the e-services and utilize their results as described by the workflow.

Knowledge of the available e-services to compare, select and invoke any service is required by the service inquirers, who want to use the service. The existing B2B exchanges and their protocols, which unite the service provider and the subscriber, are neither compatible nor interoperable with each other. New mechanisms that help in the discovery and self-organization of brokerage of e-services are needed. In our view, agent technology is one of the core technologies that will facilitate the proliferation of brokering of future e-services.

Agents can be used to exchange and represent relevant information about existing e-services, by forming a community. An e-service brokering community [4] is a self-organized virtual space consisting of a large number of agents in their dynamic environment. Within the community, highly relevant agent groups represent the e-service providers and have information about the location, quality and the interfaces provided by specific services that can be invoked by a composed workflow instance.

Apart from the discovery of e-services, we need to be able to combine different e-services into a workflow that can be again offered as a new e-service. *Workbase* is our ongoing research work, which investigates on the issues of how to automate and synchronize Internet-wide, ad-hoc

workflow [3], using e-services to create a composite service.

We contribute the concept of brokering community and its protocols, as an approach to e-services and workflow. We discuss the design and representation of e-services and show how they can be combined to form a workflow. This approach befits the current necessity to form service marketplaces that can be easily scaled to exchange-to-exchange (X2X) models.

In section 2, we give a description of *Workbase* and show how workflow can be composed of e-services. Section 3 discusses the layered architecture of the brokering agent community. Section 4 presents the components of the *Broker Based Agent Community Protocols* (BBACP). These protocols govern how agents join an e-service community, perform knowledge discovery and advertise services or capabilities. In section 5, we illustrate the prototype of a self-organizing agent community that has been implemented using JKQML [5], and discuss how it can be used for service discovery.

2. *Workbase* and e-Services

An e-service is a representation of any value-providing service that can be invoked, in the context of Internet and e-commerce. It is different from any other ordinary service by the fact, that it is web invokable. This means that any client using standard web protocols like *http* can invoke an e-service. So an e-service is a web-wrapped, service implementation. We are using e-services as the building block of our web-based workflow architecture.

Workbase – our web-based workflow engine, creates, executes, and delivers the result of a workflow definition. Workbase will compile a workflow definition into a sequence of e-services and execute the sequence, by discovering the suitable/available e-service providers. Workbase hides the complexities of reducing composite e-services, discovering and invoking basic e-services, combining results, and providing fault-tolerance.

Workbase uses the information gained from the brokering community about service providers and decides on the sequence of execution of a workflow. After finding the service and its location, Workbase uses the e-services infrastructure to invoke the service through a late binding process (the client does not know anything about the service provider before the service is actually invoked). Some e-services may not return the results immediately after invocation. For instance, some e-services may take days or weeks to finish. Workbase must be able to handle the asynchronous nature of these e-services. The brokering community and the protocols mentioned help in getting information about the location, definition, and interface of the e-service discovered and requested by Workbase.

Figure 1. depicts the core components of Workbase. A workflow engine interacts with brokering communities of e-services to bind workflow service definitions to actual e-services offered through their providers. An execution environment allows for service composition and invocation. E-services advertise their definitions and interfaces to brokering communities. The next few sections zoom into the brokering community concept and its community protocols.

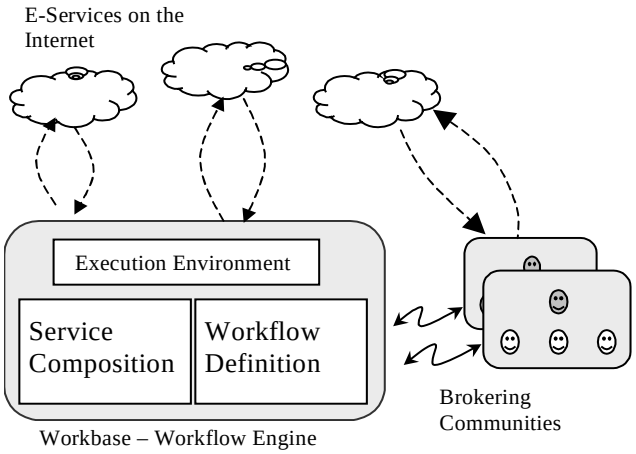

Figure 1. Overall scenario and components

3. Brokering Community

We introduce a hierarchical brokering architecture for the e-service communities. This hierarchical community acts as a broker of services between the service providers and the Workbases (service consumer) wishing to utilize e-services in their workflow. The brokering community is organized into three layers as shown in Figure 2. It consists of an *Agent Layer*, a *Brokering Layer* and a *Super Brokering Layer.*

The *Agent Layer* is the bottom layer in the community architecture hierarchy. It consists of agents or agent-based systems, which represent or have information about an e-service under a certain knowledge domain. These agents self-organize their e-service "clients" into highly relevant

communities overseen by brokers from the next layer in the hierarchy. The agents-brokers protocols are discussed in the next section. The broker of a given community guarantees the relevancy and the mutual benefits of all e-services of specified domain represented in the community by their agents. Information about e-services can be obtained by workbases by contacting one or more brokers in different communities.

The broker acts as the facilitator of the e-services between the agents and the Workbase workflow system. Agents communicate their capabilities to the brokers and the brokers use this acquired knowledge about the agents to reply to the service queries of the Workbase system. The interoperation of the agents, which represent the service providers, the Workbases, which represent the service inquirers (e.g. service consumers or business makers) with the broker, forms a single *Brokering System*. This system provides dynamic structure as agents can join or leave the community at any time. Also, a flexible co-ordination is achieved at the elementary layer.

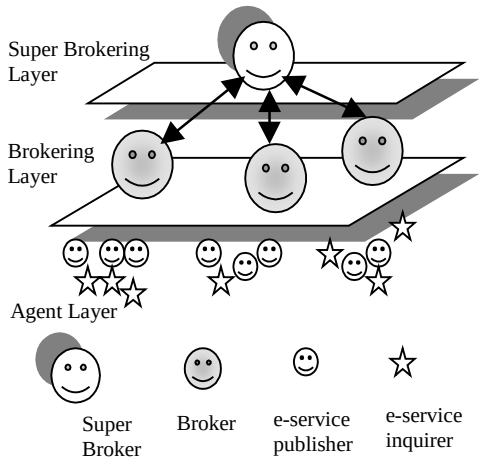

Figure 2. Hierarchical architecture of brokering community

In the next layer, which is the *Brokering layer,* the interoperation of various brokers together with a *Superbroker* form a *Multi-brokering System.* The Superbroker is a broker in the top layer supporting the knowledge sharing and discovery efforts among of the brokering community. Every broker in the brokering layer registers with the Superbroker of the community using the brokering protocols described in the next section. When a service request from a Workbase cannot be satisfied within a single brokering system, the request is sent to the other brokers within the community. The capable brokers in the community provide the requester with the information needed to obtain the service. This slowly leads to the expansion of knowledge among brokers who did not have the information before. Also, subscriptions can be made by the brokers to be notified when a capable service becomes available in the community. The Superbroker maintains the service subscriptions and notifies the subscribers, when the services become available in the community. The brokering protocol governs how brokers advertise on behalf of its agents; how service requests of the Workbases are satisfied by the broker and how brokers self-organize themselves in the community.

The *SuperBrokering Layer* is the top layer of the community. The Superbroker helps in the self-organization of the community using the *Super Brokering protocol*, which governs how superbrokers can communicate and coordinate over common ontologies between their communities.

The agent community is easily scalable to a very large size by accepting new brokers that represent a set of e-service agents and Workbases. When a broker joins or leaves the community, the Superbroker updates the knowledge among the member of the community. Superbrokers have control over load balancing within their community and support greater reliability.

Researchers have designed different organizational structures for brokering systems, such as peer-to-peer multi-brokering architecture (e.g. *InfoSleuth* [6]), the partitioned repository design in blackboard systems [7], centralized message routing design such as in *JAT Lite* [8], and the multi-facilitator mediation in OAA [9]. The hierarchical agent community is designed to meet the current requirements of scalability without any direct necessity of change in the underlying Agent Layer. The communication and interaction protocols specified in the Brokering based Agent Community Protocols are capable of avoiding single point of failure, unlike those in the *JAT Lite* and the blackboard systems. Leader election protocol may be used in the event of failure of the Superbroker to elect a new Superbroker in the community. Each broker only needs to have the information about the Superbroker it is registered with, and communicates with other brokers only upon necessity. This reduces the amount of messages flowing among the brokers within a community. Also, the community becomes self-organizing, adapting itself based on the requests for service and the availability of brokers.

In order to self-organize into communities, each broker is assumed to know a *Universal Community Ontology*, which is the fundamental ontology, used by any community. Also, *Domain ontologies* and *Community Specific Ontologies* may exist in a community [4]. Prospective community members can access and learn about these ontologies before initiating any application to join the community. The Ontologies help in the organization of the communities and its protocols.

4. Community Protocols

The Broker Based Agent Community Protocols (BBACP) is introduced to enable inter-organizational business procedures and workflow. The BBACP consists of Agent Protocol, Brokering Protocol and Super Brokering Protocol, which provide mappings from states to action in each of the community layers. The protocols were purposely kept simple.

The Agent Protocol illustrates the base level commitment and responsibility of the members to the community. It is an open protocol and space is left for negotiation after which the agents become associated with a broker.

The Brokering protocol describes a cooperative multi-brokering system, which provides the solution to interoperation among brokers in a dynamic and heterogeneous agent community. Each broker performs basic brokering functionality such as service discovery and knowledge sharing within a set of e-service agents and e-service inquirers like the Workbases, which query the brokers to complete their workflow. Individual brokers representing a set of e-service agents are allowed to advertise their business service and send capability queries to other brokers, as well as receive advertisements from other brokers. The brokering protocol regulates the joining and departure of brokers from a community. It also covers e-service knowledge discovery, sharing of acquired knowledge and information load control.

The Superbrokering protocol governs how a Superbroker communicates and coordinates in a virtual enterprise of service communities and covers the community-to-community in interaction and inter-operability. It also deals with superbroker election and maintenance of the community by the Superbroker.

With regard to e-service discovery and usage of these services in workflow by workbases, our current research focuses on the brokering protocol layer. It includes the components to regulate how brokers, which represent a set of services, join or leave the community; how knowledge discovery and exchange takes place and how advertisement of services are made. In the next subsection, we describe the community joining brokering protocol.

4.1. Community Joining Protocol

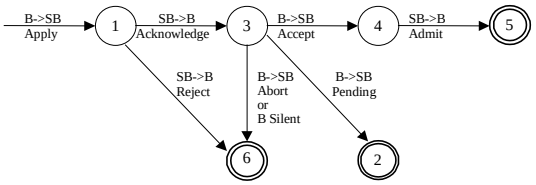

Figure 3. Finite state machine of joining protocol

A broker uses the *Joining Protocol* to join a community; it includes a logic conversation between the broker and the Superbroker. The conversation policy is a sequence of rule-based communication mapped into a finite state machine as illustrated in Figure 3. Each broker, in its application to join the community, specifies the ontology concepts it can deal with. After receiving a registration attempt (State 1), the Superbroker conducts multi-faceted membership evaluation of the applicant for credibility, possible value addition to the community, knowledge and capability relevancy among others. This evaluation results in acknowledge (State 1 –> State 3) or reject (State 1 –> State 6). The acknowledgement message from Superbroker to the applicant broker contains the domain ontology, community specific ontology and bylaws.

When the broker receives the acknowledgement from the Superbroker (State 3), it automates its behavior related to any ontology adjustment, if needed to fit into the community. The broker now autonomously makes a final decision to join the community or not based on the knowledge it can gain from the community. If a positive decision is made, an acceptance message is sent to the Superbroker (State 3 –> State 4). The finite state 5 is reached, when the Superbroker confirms the grant of membership upon receiving acceptance message. If the broker decided not to join the community, an abort message is sent to the Superbroker (State 3 –> State 6).

If a decision cannot 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. The state information in the conversation policy is maintained at the Superbroker side. During application processing in future, the Superbroker first checks the access history, following the conversation policy based on

the 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 information about the new broker in its repository. Failure of registration could encourage broker to engage in additional learning activities. The Superbroker manages the relationship between ontologies and brokers.

4.2. Community Departure Protocol

The departure protocol is used to inform the Superbroker that the broker won't be able to provide the advertised e-service(s) any more. The broker may also use this when some service is down temporarily or if the broker judges that an e-service no longer qualify to belong in the community. The leaving broker initiates the departure procedure by sending an "UNADVERTISE" message to the Superbroker. The knowledge and services available to the community are updated by sending a "TELL" message to the 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 leaving broker finally sends an "UNREGISTER" message to the Superbroker. The Superbroker makes the necessary changes in its member repository, member credit history and service quality about the leaving broker.

4.3. Knowledge Discovery and Exchange Protocol

This protocol is supposed to be used in the community by the brokers to satisfy the service queries by the Workbases (or any service inquirers). When a Workbase wants a service, it contacts the broker for the service, which finds the requested service from its brokered agent system. If the broker is not able to 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 the information about the service. This process leads to discovery and expansion of knowledge among requesting brokers. The requesting broker is able to directly negotiate with service providing broker(s). The amount of knowledge discovered is based on the brokers' objectives. The broker may request for information of a single service provider or a list of providers who satisfy the service attributes.

A broker can also request to be informed if some specific service becomes available in the community. This form of knowledge discovery is called subscription. A broker may broadcast a subscription to the community, which is stored by each member broker. If a broker (or an agent it represents) is capable of serving the subscribed service, the broker will reply to the subscription and start a conversation with the subscriber. This method of knowledge discovery will be useful for workflow, where a specific service with some constraints is required and the sequence of execution can wait for the desired service. A broker can also unsubscribe its subscriptions. Knowledge discovery protocol follows a finite state machine. If a request is satisfied and a discovery is made, the broker updates it self about the acquired capabilities. In case of an unsatisfied request, the broker may refine the request content or send a subscription to the Superbroker.

4.4. Capability Advertisement Protocol

The broker can advertise to the community about the e-services that can be provided by the agents in its brokered system. When the Superbroker receives an advertisement, it checks for any subscriptions that can be satisfied and informs the subscribers. When a member agent needs to make a change in the service advertised, it sends an unadvertise message to Superbroker and makes an update. Advertisements are basically the means of getting business for the e-service providers.

5. Implementation

Currently, we have prototyped the broker community which can be used to find the e-services required for a workflow. We have also designed the representation and invocation mechanisms for e-services, and how to convert an ordinary service into an e-service so that it can participate in a workflow.

In our implementation of the broker community, agents operate and interact in the infrastructure provided by the agent shell - JKQML [6]. All the messages used in the protocol are implemented using the KQML [10] performatives. KQML Transfer Protocol (KTP) can be used here, for TCP/IP environment. The agent community model uses the KQML manager and extends its functionality to support the brokering protocols for e-service. The implementation of the middle layer of the community protocol, which helps in service discovery for workflow, provides community-ready interface and supports the self-organizing feature of the community. Each broker object maintains three information repositories: self-capabilities repository, acquired-capabilities repository and subscription repository. The

Superbroker maintains a credential information about the member brokers and can initialize a membership evaluation procedure. All members vote on the quality of membership and submit voting results to the Superbroker. Based on the voting result, the Superbroker expunges the membership of unqualified broker(s). This ensures that only qualified brokers are present in the community. A case study was done on the broker community and the protocols for an auto-trading domain [4]. The objective of finding services related to this domain was analyzed. A set of API was provided which can be used to make self-organizing broker communities. These APIs can be overridden to support the self-organizing feature of the community of any specific domain to broker e-services [11].

An e-service wrapper is needed for any custom or legacy service application to enable it to participate in a Workbase workflow. In other words the service provider should implement some *mandatory interfaces* in order to enable itself to join workflow instances. The broker, who serves as the repository of the available services, will have information about these interfaces. The broker is an ideal place, since all the services have to register with the broker if they were to get work assigned by a workflow of a composite service. So, the broker can also act as a checkpoint to see whether the subscribed services adhere to e-services-requirement.

We plan to use existing technologies and standards that are best suited to create an infrastructure, in terms of ease-of-use, interoperability and efficiency. XML is our first choice for implementing the messaging protocol. We plan to use XML schemas and DTDs to represent service description. This means that all the service-related information stored by the broker will be XML data and suitable parsing mechanisms have to be provided. This also means that query services implemented by an e-service, such as “describe” may as well return XML data that has to be suitably parsed and utilized.

Apart from the messaging/representation protocol, we need a mechanism to actually perform service invocation on an e-service and obtain the results. This issue of service invocation has some other constraints that should be satisfied namely,

- Service invocation should be possible using HTTP.
- Dynamic invocation should be possible, meaning, there should not be a need to bind the client to the service provider at compile or build time
- There should not be any requirement constraint on the client (apart from those imposed by e-services-specification as such). That is, clients should have an interface at compile/build time to invoke the service

Upon reviewing these requirements, we decided to use SOAP [12] as the object invocation mechanism/protocol to perform service invocation.

Simple Object Access Protocol – SOAP – as the name suggests is a simple mechanism to exchange tag-based messages built on XML. We use SOAP to solve the issue of name/tag standardization we need to achieve in order to exchange messages between our entities. The advantage of using SOAP is, that some message exchange structure is already built in and we are (re)using the same to build more e-service specific information on top of it.

A sample SOAP message looks like the following:

```
<SOAP-ENV:Envelope…>
<SOAP-ENV:Body>
  <m:GetQuote xmlns:m = "www.mystockquote.com/
eservices">
      <symbol>UFL</symbol>
  </m:GetQuote>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

HTTP will be used to transport the SOAP requests and responses. So the SOAP response for the above SOAP request will look like the following

```
<SOAP-ENV:Envelope…>
<SOAP-ENV:Body>
<m:GetQuote xmlns:m="www.mystockquote.com/
eservices">
    <Price>100.01</Price>
  </m:GetQuote>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

We plan to write SOAP listeners for service providers in order to parse the incoming request and perform the corresponding invocation on the object. We currently restrict our SOAP handlers only for Java objects. In particular, the service provider apart from implementing the service will also implement the mandatory e-services interface. The SOAP handler that we will provide will do the actual job of reading the SOAP request and invoking the required implementation on the service component. This is depicted in Figure 4 below.

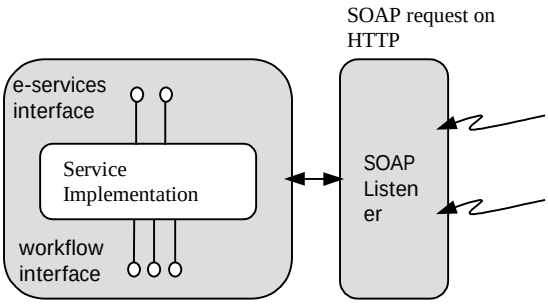

Figure 4. Service invocation on an object using SOAP and HTTP

5.1. Workflow and Dynamic Composition

The need for dynamic service composition is initiated by the need for composite service creation.

A composite service can be built by creating a workflow, which defines the sequence of execution of existing services. In addition to just defining this sequence, the workflow manager/engine also needs to take care of many workflow specific details, some of which are summarized below:

- Check the status of execution of a workflow component
- Detect occurrence of error in a workflow component and dynamically choose another provider to perform the same operations, aborting the old one
- Collect results of execution of a component and use it for rest of the workflow
- Synchronize the execution of a workflow component with the completion of other components

In order to perform the above-mentioned functionalities, the workflow engine uses the broker module, which serves as a service repository for service discovery. After finding the service and its location the engine uses the e-services infrastructure to invoke the service through late binding (by late binding we mean that the client does not know anything about the service provider before the service is actually invoked). It is also worth mentioning here that when service invocation is performed, the result of the invocation may not be returned immediately, in other words, asynchronous invocation is also possible in which case, the service provider sends an event (or message) to the requestor on completion of service request.

The sequence of operations of the workflow engine, when a composite service is invoked can be described as follows:

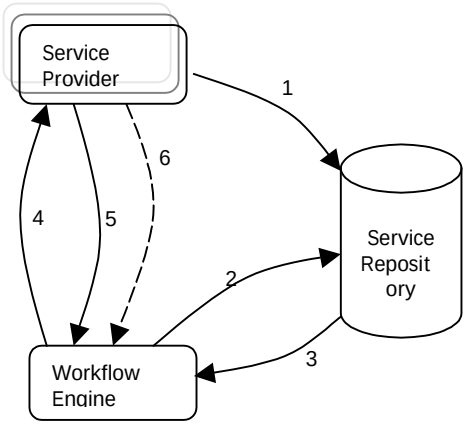

Figure 5. Protocol of service invocation

1. Businesses advertise their service details (XML DTDs) by joining a brokering community.
2. Workflow engine queries the repository to find matching services and service provider details
3. Workflow engine gets service provider details from the service repository
4. Workflow engine invokes services provided by the service provider, using e-services framework
5. Service provider acknowledges the request for service (and may return results synchronously)
6. Service invocation results can be returned asynchronously

All communication here takes place through http. Service discovery and service invocations are dynamic based on clients' request.

6. Conclusion and Future Work

In this paper, we described a hierarchically-layered agent community architecture and a set of protocols guiding interoperations across the open network for the advertisement and discovery of e-services. The brokering community concept is intended to be used as a self-organizing mechanism for related, yet autonomous e-services. The main application of such brokering communities will be future Internet-based, dynamic workflow systems. Such workflow systems will rely on the brokering communities in locating highly relevant e-

services, and in learning the interfaces required for their invocation. E-services will need to have well defined interfaces if they are to take part in such workflow systems. We have implemented the brokering community concept using JKQML and have developed a case study to demonstrate its utility [4]. We have also designed and currently implementing a SOAP-based E-service wrappers and listeners that will allow service providers to: (1) package their services (regardless of service implementation or automation details) as e-services, (2) advertise their services and self-organize them into relevant brokering communities, (3) maintain the relevancy of their e-services in the appropriate communities, and (4) participate in multiple Internet-based workflow executions by accepting binding requests from community brokers and delivering back to the workflow the rendered service.

Simple methods to determine relevancy of service information between the subscriber and the provider are used. More sophisticated algorithms are being developed to more accurately predict a broker's relevancy to the community. We are also researching the use and integration of our BBACP protocol and the emerging UDDI standard [13].

7. Acknowledgement

We would like to acknowledge Professor Stanley Su and Jie Meng for the valuable discussions on Service Brokering and Dynamic Workflow concepts.

8. References

[1] A. Dan, D. Dias, T. Nguyen, M. Sachs, H. Shaikh, R. King and S. Duri, "The Coyote Project: Framework for Multi-party E-Commerce", 7th Workshop on Electronic Commerce, Greece, 1998

[2] F. Casati, S. Llnicki, L. Jin, V. Krishnamoorthy, M. Shan, "Adaptive and Dynamic Service Composition in eFlow", HP Labs, 2000

[3] J. Meng, A. Helal, and S. Su, "An Ad-Hoc Workflow Architecture Based on Mobile Agent and Rule-Based Processing," The special session on Software Agent-Oriented Workflows, Proceedings of the International Conference on Parallel and Distributed Computing Techniques and Applications, Las Vegas, Nevada, June 2000. pp. 245-251.

[4] A. Helal, M. Wang, A. Jagatheesan, "Service-Centric Brokering in Dynamic E-Business Agent Communities," To appear in the Journal of Electronic Commerce Research (JECR), Baltzer Science Publishers. Also published as University of Florida Technical Report number: UFL-99-023.

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

[6] M. Nodine, B. Bohrer and A. Ngu, "Semantic Brokering over Dynamic Heterogeneous Data Source in InfoSleuth," *Proceedings of the International Conference on Data Engineering (ICDE)*: 358-365, 1999.

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

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

[9] G. Martin, A. Unruh and S .Urban, " An Agent Infrastructure for Knowledge Discovery and Event Detection," MCC Technical Report. Oct 1999.

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

[11] M. Wang, "Service-Centric Brokering in Dynamic e-Business Agent Communities," Master Thesis, CISE Department, University of Florida, http://www.cise.ufl.edu/~helal/~student/thesis/Mei, Dec 1999.

[12] D. Box, D. Ehnebuske, G. Kakivaya, A Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1", http://www.w3.org/TR/SOAP, May 2000

[13] UDDI, http://www.uddi.org