Market Architecture for Multi-Agent Contracting: An Internet Standards Based Approach

Leonard R. Josephs
Department of Computer Science and Engineering
University of Minnesota
josephs@cs.umn.edu

Plan B Project

Abstract

The problem of creating a software architecture suitable for the MAGNET (Multi AGent NEgotiation Testbed) system is examined and an Internet standards-based solution is presented.

1 Introduction

1.1 The Rise of Business-to-Business Electronic Commerce and Interest in Multi-Agent Markets

We have witnessed tremendous growth in the use of the Internet for electronic commerce. In the past few years alone, there has been well-publicized exponential growth in the World Wide Web (WWW) and in business-to-consumer (B2C) e-commerce. Much of the recent activity is in the rapidly expanding business-to-business (B2B) e-commerce market, with the global market expected to exceed $7.29 trillion in 2004, according to Gartner Group research.

Without a doubt, businesses are interested in leveraging the Internet and B2B e-commerce relationships in order to reduce costs, gain efficiency in core business processes, and hold strategic advantage over competitors. A recent study from Boston Consulting Group predicts productivity gains from B2B e-commerce will equal $1\% - 2\%$ of sales by 2004 and 6% by 2010.

The rising interest in B2B e-commerce has profound implications for the size and complexity of the B2B marketplace itself. The size of the B2B e-commerce marketplace is rapidly growing, both in terms of the number of participants and in terms of transaction size and volume. In fact, business-to-business hubs, which link buyers within a particular industry or across a shared need, are expected to handle as much as $1.25 trillion by 2003. Likewise, the complexity of the logistics involved in B2B transactions has grown considerably and is increasing nearly exponentially. The market for supply-chain management (SCM), customer-relationship management (CRM), and work flow automation products and services is evidence of the complexity of the logistics involved in B2B e-commerce.

A logical outcome of the enormous size and complexity of the B2B marketplace is the need to not only automate the B2B processes, but also to support intelligent decision making in B2B markets. Self-interested, autonomous software agents can be used to automate processes in a B2B marketplace, making intelligent decisions themselves or deferring to humans as appropriate (as is the case in mixed-initiative systems). Additionally, software agents may be used to model behavior over time, thereby increasing understanding of the market itself.

1.2 The MAGNET Problem

Research in modeling B2B processes and market behavior using intelligent, autonomous software agents has created the need for generalized market architectures.

Markets play an essential role in the economy, and market-based architectures are a popular choice for multiple agents (see, for instance, [3, 20, 21]).

Most market architectures limit the interactions of agents to manual negotiations, direct agent-to-agent negotiation [18, 8], or various types of auctions [22].

Researchers at the University of Minnesota created prototypes of a generalized market architecture and related agents while performing research on multi-agent contract negotiation. The resulting system was called MAGNET (Multi AGent NEgotiation Testbed). MAGNET provides support for a variety of types of transactions, from simple buying and selling of goods and services to complex multi-agent contract negotiations. In the latter case, MAGNET is designed to negotiate contracts based on temporal and precedence constraints, as well as price.

This version of the Magnet Architecture [7] has proven useful in a number of experiments, providing a simulation environment that is easily adapted to a variety of experimental purposes.

As more experiments are performed, fundamental limitations in the current MAGNET architecture are becoming more apparent. The lack of detailed architecture specifications and requirements has resulted in high code complexity and the emergence of hidden risks in scalability and usefulness of the system. In addition, new technologies have emerged that promise to enhance the utility of the system.

This paper attempts to address limitations in the current MAGNET architecture by documenting important architecture requirements and then proposing an improved architecture that satisfies those requirements. In addition, the paper will discuss the design of the improved architecture, including recommended technology choices and the initial work done to implement the new architecture.

The remainder of this paper is organized as follows: Section 2 discusses some issues associated with multiple-agent contracting, which, in turn, places high-level requirements on the system architecture. Section 3 provides a reference model describing the environment of MAGNET agents, and the basic activities and roles of agents in that environment. Section 4 briefly examines various technology options that could be used in building a suitable software architecture for the MAGNET system. Section 5 describes the architectural style needed to fulfill the high-level system requirements. Section 6 presents the technologies chosen to re-architect the MAGNET system and how they fit the architectural style and high-level requirements. Section 7 details the improved architectural model. Section 8 discusses the specifics of the implementation work performed. Finally, Section 9 summarizes the results of the analysis, and outlines future plans and open problems.

2 Architectural Requirements

The development of high-level requirements for an architecture is important for measurement of the success of the architectural design and for understanding the basis of design decisions [1]. Successful requirements address the needs of project stakeholders. The business case for the project and the demands placed on the project by the project's stakeholders must be carefully considered in order to create a successful architecture [1].

The business case was outlined in section 1.1 of this paper. Essentially, the architecture must support research activity in market-supported agent negotiation and contracting.

2.1 Interested Parties

The following stakeholders need to be considered in order to assure the success of the improved MAGNET architecture:

2.1.1 Primary Research Team

The MAGNET project is a joint effort between researchers at the University of Minnesota and at DePaul University. Researchers on both campuses are actively writing code to test new research ideas, so flexibility is key. In addition, the team does not want to lose any functionality in the system so that experiments may be re-run and earlier work extended. Recent software development at Minnesota has centered around agent design, bid evaluation, graphic user interfaces to agents, protocols, and architecture while development at DePaul has involved markets, exchanges, and server design. Therefore, agents need well-defined but loosely-coupled interaction with the market and associated server or servers. Finally, it would be nice for each team to able to run the system at either campus and easily access it from the other.

2.1.2 Research Community

Another goal of MAGNET would be to allow the larger research community (others interested in multi-agent negotiations using a market framework) to extend work on the MAGNET system. In particular, it seems plausible that other types of agents and/or negotiation protocols could be used within the MAGNET architecture.

2.1.3 National Science Foundation (NSF)

The MAGNET project has partial funding from the NSF. The NSF has an interest in seeing that MAGNET research benefits society, provides educational opportunities, and that NSF funds are used wisely towards these goals. Therefore, cost is an issue as funds are limited and are intended to be directed towards research and education, rather than expensive software tools and platforms.

2.1.4 Business Community

The business community has significant interest in learning from MAGNET and using MAGNET-related technology to automate business practices. In particular, MAGNET technology could be used for Supply-Chain Management, automated contracting, and other types of business-to-business e-commerce. Therefore, MAGNET must support transactions between agents that have qualities such as security, transactional consistency, and authentication, since similar properties are required in traditional (non-automated) business transactions. Details of using MAGNET for Supply-Chain Management may be found in [5]. The use of MAGNET in automated contracting is discussed in [4].

2.2 Driving Forces and Derived Requirements

2.2.1 Breadth of Agent Participation (BAP)

Maximizing the number and variety of agents that may be supported in the MAGNET architecture should be a major goal of the system as a whole. Unintentional or unnecessary constraints on the pool of agents that may participate in the market undermines the usefulness of the market as a whole and lead to market inefficiencies. The restrictions on allowable agents in a market directly impacts the performance of the market. Allowing agents that are spread out across the Internet to interact in the MAGNET environment is key to gaining wider acceptance as a research tool and important in simulating larger, more diverse markets. Therefore, the following requirements may be derived:

BAP-1: Open Communications Agents should be able to participate in the MAGNET market over the open Internet. This implies the need for information to flow easily across network boundaries (such as firewalls) by using widely-accepted Internet standards and protocols.

BAP-2: Heterogeneous Agents Agent implementation should not be dictated by the structure of the MAGNET architecture. In particular, agents shouldn't have to be tightly coupled to the MAGNET market or session.

BAP-3: Agent Platform Independence Another goal of the MAGNET Architecture should be to minimize the effect of heterogeneous platforms and computing environments on the ability of agents to participate in MAGNET markets. Agents should not have to be written in a particular language or run on a specific platform.

BAP-4: MAGNET Market Support MAGNET marketplaces and associated servers should be able to be hosted on a variety of platforms. At a minimum, MAGNET should be operable in the UNIX (Solaris and Linux) and Windows environments since they are in use by the MAGNET research team.

2.2.2 Flexibility and Adaptability (FA)

As MAGNET is intended to serve the research community, it is important that the MAGNET Architecture be flexible enough to allow for a variety of experiments to be performed. Agents and market components will likely need to be adapted in order to model different problem domains; the MAGNET Architecture should be supportive of change.

FA-1: Extensible Protocol The MAGNET system is a research vehicle; agent negotiation protocols are under research as well and therefore are subject to change. Messages and parts of messages that are not completely understood by agents or the MAGNET system should be recorded or reported and not cause system failure.

FA-2: Open Protocol The protocol can not be opaque to the MAGNET session. Agent communications should be be open to the MAGNET system for inspection. For instance, the protocol needs to be visible to MAGNET server so that the servers can manage the distribution of the protocol elements (such as the RFQ) to registered agents who should be informed and so that MAGNET servers may manage bid timeouts. As is discussed in detail in [7], a MAGNET server can act as a trusted intermediary, providing important protections to participating agents, and this requires MAGNET servers and the agents to have access to the negotiation protocol.

FA-3: Runtime Flexibility The MAGNET architecture should lend itself to the use of design patterns that offer runtime flexibility so that agents, markets, and the like can be easily configured with minimal impact to the source code. The use of configuration parameters, modules, and factories is encouraged since this allows experiments to be quickly configured, run, and reproduced in the MAGNET environment. The details of the factory pattern in object-oriented design is described fully in [10].

FA-4: Scalability Although extremely high numbers of MAGNET sessions and participating agents is unlikely in a research prototype, care should be taken to create a MAGNET architecture that doesn't place arbitrary limits on scalability. Poor scalability could limit the interest in adopting MAGNET technologies to commercial use, since scalability is important to the business community and other interested parties for real-world applications.

2.2.3 System Integrity (SI)

System integrity is key to the acceptance of the MAGNET system both as a research tool (results must be consistent and reproducible) and as technology valuable to the business community.

SI-1: Agent Identity Agent identity needs to be persistent and traceable to a responsible person or organization.

SI-2: Security Basic security mechanisms will be required for the use of MAGNET on the Internet or other open networks since MAGNET may be used to model or automate real-world business processes. Agent-to-market authentication, privacy of message data, message integrity, and possibly audit and non-repudiation can be provided by a basic security mechanism. The architecture should provide a means of basic security that requires minimal impact to application code. Note that user-to-agent authentication is the responsibility of the agent and therefore not addressed by the MAGNET architecture.

SI-3: Persistent context The MAGNET architecture needs to support the notion of a persistent context in which negotiation takes place. Agents should be able to start, stop, disconnect, and connect without losing identity or context. In particular, shutting down an agent should not allow the agent to repudiate commitments or result in the loss of messages intended for the agent.

2.2.4 Software Cost and Maintenance (SCM)

As mentioned in section 2.1.3, there are limited funds and resources available to the MAGNET project, especially for the use of improving the underlying architecture. Cost and maintenance requirements of the architecture must be considered in order allow scarce resources to be focused on research needs.

SCM-1: Software Reuse It is important that the improvements to the MAGNET architecture make as much use of pre-existing MAGNET source code as practical. Code reuse and especially design reuse can cut the cost and effort associated with improving the MAGNET architecture. Numerous case studies and antidotal evidence supporting the ability of design and code reuse to cut costs can be found in [23].

SCM-2: Low Cost The MAGNET architecture should be affordable to implement; in particular, care should be taken to avoid expensive technologies and software solutions.

SCM-3: Active Support The underlying technologies used for the MAGNET architecture should be actively supported. This is so that improvements and new developments may be incorporated into MAGNET and so that defects found in the underlying technologies may be corrected with a minimum of effort from the MAGNET team and interested parties.

3 Reference Model

A reference model is defined in [1] as "A division of functionality together with data flow between the pieces". The reference model must therefore address the major components of the MAGNET system and the interactions between them.

Some basic architecture work that was done to identify components and interactions in the MAGNET system may be found in [7]. The reference model is based off this work.

3.1 Introduction

The MAGNET reference model is taken directly from [7] and contains agents, exchanges, markets, and the protocol used to interact between these components. A conceptual view of the components is shown in Figure 1.

MAGNET provides an agent the ability to use market mechanisms to discover and commit the resources needed to achieve its goals. The assumption is that agents are heterogeneous and self-interested and typically act on behalf of entities who have different goals and different notions of utility. Agents may fulfill the role of customer or supplier with respect in the MAGNET reference model, as shown in Figure 1.

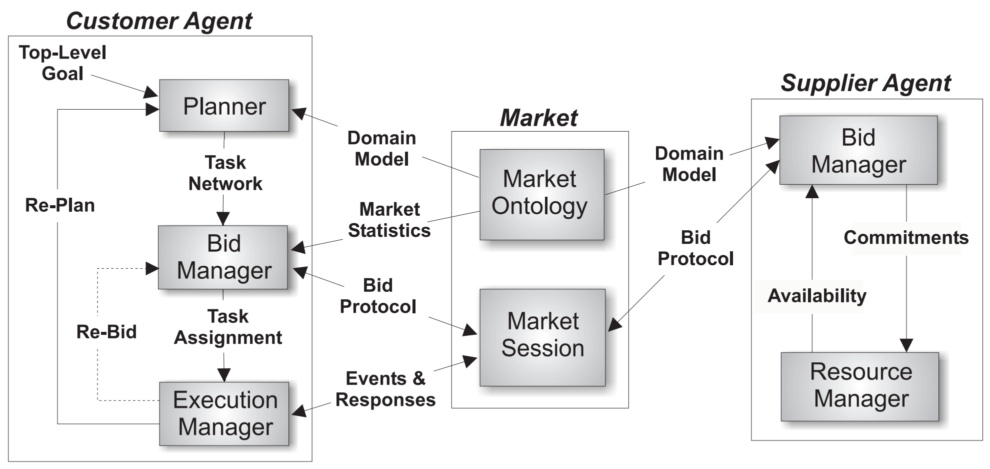


Figure 1: The MAGNET reference model

3.2 Customer Agent

Customer agents plan and then pursue their goals by formulating and presenting Requests for Quotations (RFQs) to supplier agents through a market infrastructure [6]. Customer agents next evaluate the bids received from supplier agents, and award bids to selected supplier agents. Finally, customer agents may monitor execution of the tasks specified in the awarded bids. Customer agents attempt to satisfy their goals for the least net cost, where cost factors can include not only bid prices, but also goal completion time and risk factors. More precisely, these agents are attempting to maximize the utility function of some user, as discussed in detail in [4].

3.3 Supplier Agent

Supplier agents attempt to maximize the value of the resources under their control by submitting bids in response to customer RFQs, specifying what tasks they are able to undertake, when they are available to perform those tasks, and at what price. See [4] for more details of supplier agent goals.

3.4 Exchange

An exchange is a collection of domain-specific markets in which goods and services are traded, along with some generic services required by all markets [7]. Possible services in an exchange include an identity verification service or a Better Business Bureau that can provide information about the reliability of other agents based on past performance. The exchange is a network-accessible resource that supports a set of markets and common services. Agents can use the exchange to find markets to participate in. An example exchange is depicted in Figure 2.

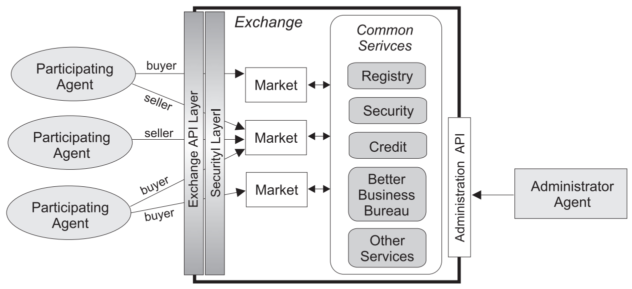


Figure 2: The Structure of an Exchange

3.5 Market

Each *Market* within an exchange is a forum for commerce in a particular commodity or business area [7, 4]. Each market includes a set of domain-specific services and facilities, as shown in Figure 3, and each market draws upon the common services of the exchange.

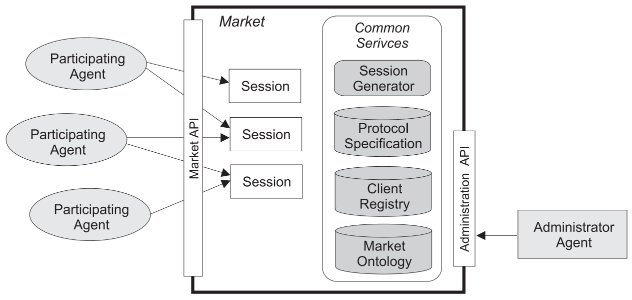


Figure 3: The Structure of a Market within the Exchange

The market contains an *Ontology* that describes the types of tasks or goods that the market deals in. Each description not only describes the item, but also contains statistics, including details like the number of suppliers that typically will bid on the item, and how long the task typically takes [4]. The market also keeps a *Registry* of suppliers that have expressed an interest in participating in market activities, and maintains performance statistics that customers can use in their decision processes.

3.6 Market Sessions

An important component of each market is a set of current sessions in which the actual agent interactions occur. A *market session* is the vehicle through which market services are delivered dynamically to participating agents. It serves as an encapsulation for a transaction in the market, as well as a persistent repository for the current state of the transaction, throughout the life of the contract [4, 7].

3.7 Protocol and Typical Data Flow

3.7.1 Agent-Market Interaction

Before Customer-Supplier Interaction may occur, both types of agent must communicate with the exchange in order to find the markets that they may participate in. After the exchange has returned a simple list of the markets available to the agent, the agent chooses the market to participate in and registers with the market. Finally, the agent requests ontology information (the types of goods or tasks the market deals in) which is returned as a set of information objects.

3.7.2 Customer-Supplier Interaction

The bidding interaction between customer and supplier agents starts with a Request for Quotes (RFQ) issued by the customer, followed by a set of bids submitted by interested suppliers, and concludes with a set of bid awards issued by the customer. After contracts are awarded, the execution phase starts. The exact protocol for the execution phase is considered an open issue.

A sequence diagram showing customer-supplier interaction is depicted in Figure 4.

- The customer agent issues an RFQ to the market for consideration by suppliers. The RFQ specifies a task network, which includes a specification of each task, and a set of precedence relations among tasks. For each task, a time window is specified giving the earliest time the task can start and the latest time the task can end.
- Suppliers may respond to an RFQ with a bid on a task or tasks. Bids may specify individual or combinations of tasks with a single price or individual prices. Bids also specify time information. A supplier's bid includes a price for the task(s), a portion of the price required to be paid as a non-refundable deposit at the time the bid is awarded, an estimated duration for the task(s), and a time window within which the task(s) can be started.
- When the customer awards a bid, it must pay to the supplier the deposit and specify the actual time, within the supplier's specified time window, at which it wishes to begin the task.

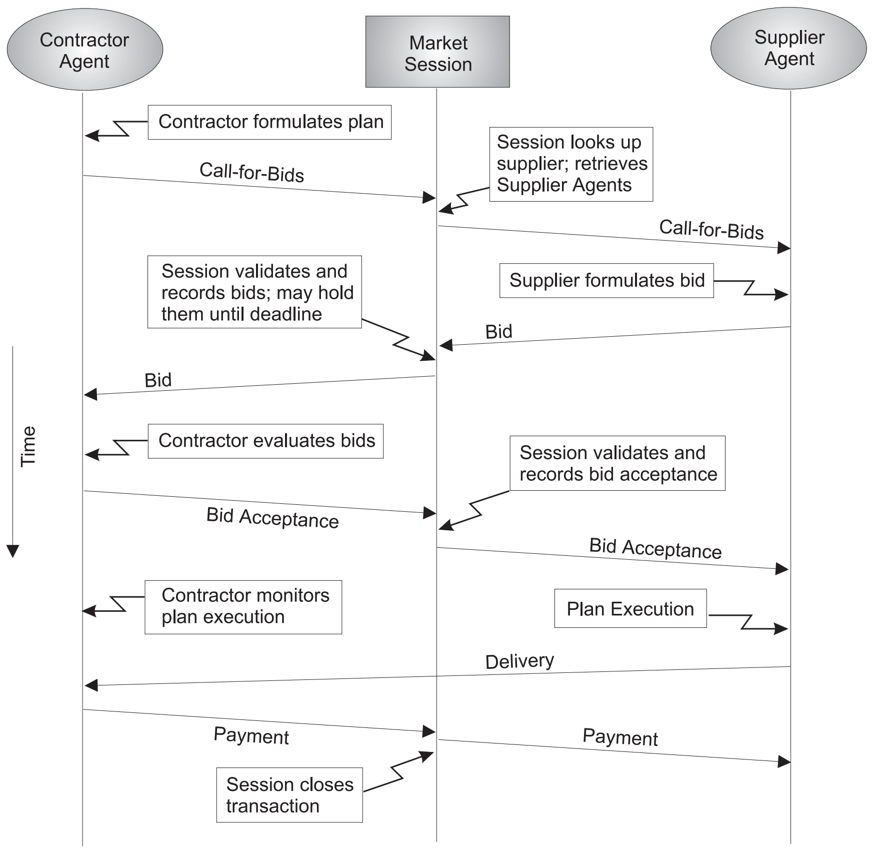

Contractor Agent
Market Session
Supplier Agent
Contractor formulates plan
Session looks up supplier; retrieves Supplier Agents
Call-for-Bids
Call-for-Bids
Session validates and records bids; may hold them until deadline
Supplier formulates bid
Bid
Bid
Time
Contractor evaluates bids
Session validates and records bid acceptance
Bid Acceptance
Bid Acceptance
Contractor monitors plan execution
Plan Execution
Delivery
Payment
Payment
Session closes transaction

Figure 4: Customer-Supplier Interaction in a Typical Contracting Market Session

- When the supplier completes a task, the customer must pay the remainder of the price, beyond the deposit, as specified in the awarded bid.
- If the supplier fails to complete a task, the price is forfeit and the deposit must be returned to the customer. A penalty may also be levied for non-performance, but we ignore this complication at this point.

4 Technology Options

The intent of this section is to briefly examine current technologies that could be used as the basis of the MAGNET framework. The emphasis is on technologies that may be used for the framework supporting agent participation rather than technologies for the agents themselves, since it is expected that agents will be implemented using a variety of technologies (see requirements [BAP-2: Heterogeneous Agents] and [BAP-3: Agent Platform Independence]). Put another way, the technologies considered in the rest of this section are capable of supporting communication between remote agents and the market and can be used to build the protocol, exchange, market, and session components discussed in Section 3.

4.1 Client-Server Technologies

4.1.1 Socket Communications

Simple socket communications could be used between agents and the MAGNET market. This would involve writing a server that fields requests from agents. A concern with this approach is the format of the protocol used for client (agent) to server communications. The implementation decides the protocol and message format, meaning that unless standards-based protocols are used (for example, HTTP or SMTP), the protocol will be opaque to non-MAGNET systems, preventing the crossing of network boundaries (see the requirement [BAP-1: Open Communications]).

Both the agent and server must be written to understand the protocol and message format. This leads to tight coupling between components and "brittle" systems, where a simple change in a protocol element, server version, or client version results in the need to update all components in the system.

Secure Sockets Layer (SSL) can provide "wire" security for sockets-based communication.

It should be noted, however, that socket communications underlie all the discussed network technologies in this paper and are in a sense the “lowest common denominator”. As such, it is always possible (with potentially a great deal of effort) for a component on a platform that doesn’t support the technology discussed to participate using sockets. The amount of effort involved to fully implement the protocols and related supporting services of these technologies makes it unrealistic to use sockets in almost all cases.

4.2 Distributed Object Technologies

Distributed Object Technologies allow components to interact at a high level by making method invocations on remote objects as if they were method invocations on a local object within the memory space of the invoker. See Section 5.1.2 for a discussion of distributed object architecture. There are three major distributed object systems in use today: CORBA, DCOM, and Java RMI. Enterprise JavaBeans are also discussed.

4.2.1 Common Object Request Broker Architecture (CORBA)

The Common Object Request Broker Architecture (CORBA) is a distributed object specification for achieving interoperability between distributed computing nodes [12]. CORBA 1.1 was introduced by the Object Management Group (`http://www.omg.org`) in 1991. This specification defined an Interface Definition Language (IDL), an Application Programming Interface (API), and an Object Request Broker (ORB) that provide the framework for distribute objects to interact. Essentially, the ORB acts as a bus connecting the objects, allowing objects to make method calls on each other [16]. CORBA 2.0 specifies an Internet Inter-ORB Protocol for communications between ORBs supplied by different vendors and distributed over the Internet. In addition, CORBA defines a number of services, such as object activation, security, transactions, and object discovery (naming, trader services) that allow rich object interactions. IDL supports multiple language bindings (mapping of CORBA to and from language-specific constructs), including Java, C, C++, Lisp, COBOL, Smalltalk, Ada, Python, and IDLScript. Therefore, the requirement [BAP-3: Agent Platform Independence] is satisfied, since IDL can be compiled on a variety of platforms and languages. See the CORBA Specification [13] for additional details.

Objects in CORBA interact across the Internet using IIOP, a binary protocol. Unfortunately, many enterprises do not allow IIOP across network boundaries since IIOP is not easily inspected for malicious content. Hence, requirement [BAP-1: Open Communications] is not satisfied.

Security is supported in CORBA both at the service level (via the security service) and "on the wire" (via SEC-IIOP, or SSL with IIOP). Persistence may be supported using the persistence service. Therefore, MAGNET requirements [SI-1, SI-2, and SI-3] are all potentially satisfied. Unfortunately, not all CORBA vendors support all the CORBA services, including the security and persistence services. Therefore, as a practical matter, there is a danger of becoming tied to a particular CORBA ORB or vendor and losing some interoperability..

CORBA implementations are available from commercial vendors such as Iona and Borland. Numerous free CORBA ORB and implementations exist; some free implementations include MICO, JacORB and Orbacus. Therefore, the requirements [SCM-2: Low Cost] and [SCM-3: Active Support] are satisfied as well.

4.2.2 Distributed Component Object Model (DCOM)

Distributed objects can also be created using Microsoft's Distributed Component Object Model. DCOM uses a object remote procedure call (ORPC) on top of Microsoft's distributed computing environment to allow interaction with remote objects as if they are local. DCOM ORPC is a binary protocol, like CORBA's IIOP, and as such tends to be blocked at Internet firewalls. Thus requirement [BAP-1: Open Communications] is not supported.

Like CORBA, DCOM allows components to be implemented in a variety of languages, including C++ and Visual Basic. DCOM is a purely Microsoft technology; all objects in the system must be implemented using Microsoft's technology in order to participate in the DCOM environment directly. DCOM does not satisfy requirements [BAP-2, BAP-3, or BAP-4] because of the need to use Microsoft solutions for all components. This situation is referred to as the "Vendor Lock-In" architecture AntiPattern in [2] and tends to have negative effects on the cost and ability to maintain the software. In addition, open standards are difficult to maintain in such a situation.

4.2.3 Java Remote Method Invocation (RMI)

Java Remote Method Invocation (RMI) is the distributed object system built into the core Java environment [9]. Like CORBA and DCOM, RMI transmits method invocation requests in a binary format. Hence, requirement [BP-1: Open Communications] is not satisfied.

In addition, RMI is a Java-only protocol; distributed objects must be implemented in Java, although Java provides ways around this limitation.

- Java provides a means to access objects written in other languages through the Java Native Interface (JNI). Therefore, objects written in other languages may use JNI to access a proxy or wrapper object that speaks Java RMI. This approach works for all platforms that support a Java Virtual Machine (JVM).

- Newer versions of Java RMI have been enhanced with the ability to use IIOP as the transport protocol for RMI requests; therefore, non-Java objects may use CORBA to communicate with Java RMI objects.

Using either means described above, it is possible to support requirements [BAP-2: Heterogeneous Agents] and [BAP-3: Agent Platform Independence].

Java technology is free, alleviating cost concerns. Software reuse is maximized, since the current MAGNET system is implemented in Java and Java is actively supported by Sun Microsystems, among others. Therefore, requirements [SCM-1, SCM-2, and SCM-3] are also satisfied.

4.3 Web-based Technologies

Web-based technologies use the HyperText Transfer Protocol (HTTP) as a means of communication between remote components. Therefore, all of these technologies use open, text-based communication that will satisfy requirement [BAP-1: Open Communications].

4.3.1 Common Gateway Interface (CGI)

The Common Gateway Interface (CGI) provides a means of executing programs from a web server. In essence, the web server forwards requests for specific resources to an external program for processing. The output of this program is then sent back to the client in place of a static file.

The use of CGI processing allows data to flow between remote clients over HTTP, an Internet standard, thereby supporting requirement [BAP-1: Open Communications]. An important problem is that while the data transport mechanism is standard, the data encoding used is not. Protocol data has to be encoded into HTML "forms" or parameters, leading to a large amount of code for parsing requests and encoding responses in the CGI program. This tight "coupling" of protocol to the implementation is similar to the client-server protocol issues discussed above and leads to difficulty in supporting the [BAP-2: Heterogeneous Agents], [FA-1: Extensible Protocol], and [SCM-1: Software Reuse] requirements, since the protocol encoding scheme is not standardized.

CGI programs may be written in almost any language, so requirement [BAP-4: MAGNET Market Support] is satisfied by the CGI approach.

The life-cycle for CGI programs can result in great demand on web servers since new processes are often spawned for CGI programs. This approach is not considered to be as scalable as the plugin and servlet approaches (which better support requirement [FA-4: Scalability]) [14].

4.3.2 Web Server Plugins

Web server companies (such as Netscape and Microsoft) have created proprietary extension APIs or plugins. Plugins have the ability to change or extend the web server's functionality by calling linked-in classes written in C or C++. The approach is extremely fast but causes security issues, since a bug in a plugin can cause the entire web server to crash[14]. Therefore, plugins are a poor choice for market infrastructure, since malicious agents have the ability to use bugs in plugins to subvert or disable the entire market. In addition, the plugin approach is highly-vendor specific – market code will vary depending on the web server to be supported (thereby failing requirement [BAP-4: MAGNET Market Support] and [SCM-1: Software Reuse]).

As with the CGI approach, a protocol data-encoding format would need to be implemented between the client and plugin. As discussed in Section 4.3.1, requirements [BAP-2: Heterogeneous Agents], [FA-1: Extensible Protocol], and [SCM-1: Software Reuse] are poorly supported.

4.3.3 Java Enterprise Edition Technologies

Servlets and JSP Java supports the notion of a "servlet" which is a generic server extension. The most common form of servlet is the HTTP servlet, which is used to extend web server functionality. Servlets are handled by separate threads within a web server process but do not share the security concerns of plugins since they are run within a Java Virtual Machine [14]. Servlets are more efficient than CGI. Additionally, servlets are supported in most major web server platforms and are Java programs, so servlets are a relatively platform-independent solution that satisfies requirement [BAP-4: MAGNET Market Support].

As with the CGI and plugin approaches, a protocol data-encoding scheme would need to be implemented, in this case between the agents and servlets. As discussed in Section 4.3.1, requirements [BAP-2: Heterogeneous Agents], [FA-1: Extensible Protocol], and [SCM-1: Software Reuse] are therefore poorly supported by this approach.

Java Server Pages (JSPs) are HTML pages that contain snippets of Java code. When a JSP is fetched by a web server on behalf of a client, the JSP is complied into a servlet class. Therefore, JSPs offer much the same functionality as servlets [14] and satisfy the same requirements as the servlet approach..

Enterprise JavaBeans and Application Servers Enterprise JavaBeans (EJB) are Java components (objects following the JavaBean specification) that communicate using Java RMI and that interact in the EJB environment [9]. The EJB environment (the EJB container)is provided by an Application Server, which typically provides a web container as well. The EJB container provides a number of distributed object services, including object lookup (via JNDI), persistence (via Entity Beans), and transaction processing. Therefore, EJBs are ideal for web-enabled applications, since the web container provides web-based access to the application, and the business logic may be placed in distributed components across the enterprise that may be transactional and persistent [9].

Note that EJBs communicate with each other using RMI; therefore, EJBs don't communicate across the Internet well. EJBs communicate well within the enterprise, but need to resort to web-based communication in order to communicate across the Internet (to satisfy requirement [BAP-1: Open Communications]).

Enterprise JavaBeans are a part of Java 2 Enterprise Edition (J2EE), which is freely available. The J2EE specification [19] provides more details about EJBs and the services available to EJBs in the J2EE environment. Application servers for deploying EJBs are available from commercial vendors (BEA's Weblogic or IBM's WebSphere products). Free versions are also available (for example, JBoss or Enhydra). Therefore, the [SCM-1, SCM-2, and SCM-3] requirements are satisfied.

4.3.4 Active Server Pages

Active Server Pages (ASP) is a Microsoft technology for generating dynamic web content using HTML pages containing snippets of embedded code (usually VBScript or JScript). The code snippets are read and executed by the web server before the page is sent to the client [14]. ASP is optimized for generating small portions of dynamic pages [14].

Support for ASP is built into Microsoft's Web Server (IIS); third-party products (Sun's Chili!Soft for example) allow it to work with other servers at significant cost. Therefore, Vendor Lock-In [2] and cost are issues (failing requirement [SCM-2: Low Cost]) as is platform support (requirement [BAP-4: MAGNET Market Support]).

4.3.5 HTTP-based SOAP Service Providers

The Simple Object Access Protocol (SOAP) is a text-based wire protocol that uses Internet standards (HTTP) for data transport and the eXentsible Markup Language (XML) for data encoding [15]. SOAP is a standard controlled by the World Wide Web Consortium (W3C) and backed by several industry giants including IBM and Microsoft (the SOAP specification may be found at `http://www.w3.org`). The use of the HTTP Internet standard means that data is transported across network boundaries easily (satisfying requirement [BAP-1: Open Communications]).

A SOAP web service receives a service request as an SOAP-encoded message (an XML document) and returns an XML document as a response. The SOAP framework takes care of data encoding, decoding, and execution of the proper method on the service object. Therefore, agent and market implementation is independent of data encoding issues and requirements [FA-1: Extensible Protocol], [FA-2: Open Protocol], and [SCM-1: Software Reuse] are supported.

Apache and Microsoft have free SOAP toolkits for sending and receiving SOAP messages, and more vendors are planning SOAP support in their products. Therefore the SOAP approach meets requirements [SCM-2: Low Cost] and [SCM-3: Active Support].

4.4 Messaging Technologies

4.4.1 Message-Oriented Middleware

Message-Oriented Middleware (MOM) systems provide a means of asynchronous communication between components. Components may exist on separate and heterogeneous platforms. Components use the middleware to send and receive messages; the middleware typically provides implements message encoding, decoding, and message queuing. Examples include MQSeries from IBM and Tibco's Rendezvous [17]. Other vendors include SilverStream, Oracle, and BEA. "Vendor Lock-In" [2] is a particular danger, since these propriety solutions do not necessarily interoperate. In addition, all the examples investigated were expensive enough to not merit consideration for MAGNET (failure of requirement [SCM-2: Low Cost]).

4.4.2 Java Messaging Service (JMS)

Java Messaging Service provides a common interface over various Message-Oriented Middleware implementations. Essentially, a Java developer can write code that uses JMS for messaging and not worry about which particular message provider will be delivering the messages. Note that JMS is subject to the same limitations as the Message-Oriented Middleware it "wraps", including cost.

4.4.3 SOAP, XML, and e-mail

A pervasive, stands-based messaging scheme exists today – e-mail. E-mail protocols, such as SMTP can be used as a message transport and XML can be used to represent the message data. A variety of specialized XML message formats exist for e-business message exchange. The SOAP protocol has become a standard format and can be used to encode service request parameters in XML. Apache SOAP, mentioned in section 4.3.5, can also be configured to work as a messaging service as well as a web service.

5 Architectural Style

An architectural style is defined in [1] as “a description of component types and a pattern of their runtime control and/or data transfer”. A style can be thought of as a group of constraints on an architecture; one example given in [1] is the client-server architectural style. The candidate technologies discussed in section 4 can be generally grouped into the following styles: client-server, distributed objects, web services, and message-oriented middleware.

5.1 Candidate Architectural Styles

5.1.1 Client-Server

The client-server style implies that multiple clients exist and communicate with a server using a shared protocol. Typically, this results in a tightly-coupled system, often with specific algorithms to encode and decode protocol elements implemented in the clients and server. Therefore, requirements [BAP-1: Open Communications], [BAP-2: Heterogenous Agents], and [FA-1: Extensible Protocol] are likely not to be satisfied. In addition, there is waning interest in this architectural style; this is primarily due to a lack of scalability (requirement [FA-4: Scalability] is not satisfied).

5.1.2 Distributed Objects

In the 1990’s, the object-oriented community pushed for the development of an Object RPC (Remote Procedure Call) that would link objects to communication protocols [12]. This led to “distributed object” middleware that could locate and instantiate a target object in a “server” process. To the application programmer, a method invocation on a remote object “looks” like an invocation on a local object. The distributed object style is highly scalable, as objects may be instantiated on many nodes in the system without regard to location or the server processing the request. CORBA, DCOM, and Java RMI are the dominant distributed object technologies in industry today (see section 4 for a discussion of these technologies). All three technologies allow object implementation to occur in multiple languages (see section 4) and therefore support the [BAP-3: Agent Platform Independence] requirement to some degree. Security is much easier to obtain than in traditional client-server systems, as security is often part of the distributed object framework.

A problem with these technologies is that communication takes place between objects in a binary format; this causes difficultly in operating across the Internet since most firewalls and network proxies are not configured to pass this sort of traffic. In addition, these binary formats are not completely interoperable with each other, sometimes requiring special bridging software at additional cost and complexity. For example, bridges exist for interfacing CORBA's Inter-ORB Protocol (IIOP) to DCOM's Object Remote Procedure Call (ORPC). Therefore, the [BAP-1: Open Communications] requirement is not well supported, making this style ill-suited for agent-to-market communications across the Internet.

5.1.3 Web Services

Web services were created specifically to address the needs of dynamic e-business across the Internet. The web service architectural style is essentially a Service-Oriented Architecture (SOA) style, implemented with Internet standards such as the eXtensible Markup Language and the HyperText Transport Protocol (HTTP). Therefore, the [BAP-1: Open Communications] requirement is satisfied by this architectural style.

The Service-Oriented Architecture, as described in [11], consists of:

- Service Providers that provide a service interface for a software asset. A service provider node can represent the services of a business entity or it can simply be the service interface for a reusable subsystem.
- Service Requesters that discovers and invoke other software services to provide a business solution.
- The service broker, which acts as a repository for software interfaces that are published by service providers.

Note that this style is scalable (satisfies [FA-4: Scalability]) since Service Providers and brokers can be spread over a number of network nodes.

5.1.4 Messaging

Message-Oriented Middleware (MOM) systems provide a means of asynchronous communication by providing a framework for messages to be passed between software components. Examples include MQSeries from IBM, Tibco's Rendezvous, and Java Messaging System (JMS) providers [17]. The primary advantage to messaging systems is that components may be loosely coupled; agents interacting with a server through messaging may continue work without waiting for a response. Therefore, the [BAP-2: Heterogenous Agents] may be easily achieved. Scalability [FA-4] may also be achieved through the use of message routers.

Proprietary MOM systems can be costly (failure to meet [SCM-2: Low Cost]) and may not easily provide messaging across the Internet due to the use of non-standard protocols (failure to meet [BAP-1: Open Communications]). A recent trend towards XML-based messaging using standard protocols, such as POP3 (Post Office Protocol) and SMTP (Simple Mail Transfer Protocol), alleviates these concerns. See [17] for an introduction to XML in Messaging and an example XML message broker.

5.2 Selected Architectural Style: Web Services and Messaging

It is expected that MAGNET component interactions will require both asynchronous and synchronous communications. Therefore, an ideal approach would be to take the Web Services architecture style and add messaging to it. In essence, the Service-Oriented Architectural (SOA) style will be used, but in addition to the web technology added to the SOA by web services, a messaging technology will be added as well. For example, an agent can be modeled as a service requester, requesting some service from a provider (which is essentially an interface to the market). In the case of the service requested, the agent looks up the service and discovers that it may participate using a web-based synchronous approach (HTTP) or a message-based approach using e-mail. The agent then chooses the service, making a request on it and receiving the reply either synchronously or asynchronously depending on the service selected.

6 Technology Choices

The web service and messaging architectural style was chosen in Section 5.2. This section addresses the technologies chosen to supply the web services and messaging needed for the selected architectural style. The technology candidates were originally presented in Section 4.

6.1 Web Services

6.1.1 SOAP Service Provider: Apache SOAP

The Apache SOAP service provider was easily chosen over Microsoft's SOAP services for the following reasons:

- Apache SOAP provides a Java API; Microsoft does not. The MAGNET system is implemented entirely in Java, so integrating the Microsoft solution would be significantly harder than integrating Apache SOAP. Apache SOAP does a better job of supporting requirement [SCM-1: Software Reuse] with respect to the MAGNET system.

- Apache SOAP provides integration with far more web servers than Microsoft SOAP. Apache's SOAP service provider runs as a servlet or JSP in more web servers; Microsoft's SOAP toolkit uses Web Server Plugins, Active Server Pages (ASP), and Dynamically Linked Libraries (DLLs) to provide SOAP Services, thereby constraining the Microsoft SOAP service provider to Microsoft's web server and platform (see `http://msdn.microsoft.com/library/en-us/soap/htm/soap_overview_3drm.asp` for details). Microsoft SOAP's service provider does a poor job of supporting requirement [BAP-4: Magnet Market Support].

- Apache SOAP service requesters (agents/clients) use Java; the Microsoft toolkit provides a VBScript API for service requesters. The Microsoft SOAP toolkit is far more limited in the platforms supported by the toolkit and therefore provides little support for requirement [BAP-3: Agent Platform Independence]. Note that agents in other languages can still request a SOAP service from either service provider (with significantly more effort) by composing the XML request, posting it using HTTP over a TCP socket, and parsing the XML response.

- Apache SOAP has messaging support (addressed later in this section); Microsoft SOAP does not.

6.1.2 Web Server: Tomcat

There are a number of web servers that support Java servlets (and could therefore support Apache SOAP). The Tomcat web server, however, is developed by Apache as an open source project specifically for the execution of servlets and is freely available from `http://jakarta.apache.org/tomcat/index.html`. Sun has adopted Tomcat as the official reference implementation for Servlets and JSPs. Finally, as Apache has control over the open source implementation of both Apache SOAP and Tomcat, the combination has been used together extensively and documentation exists for using the two technologies together.

6.1.3 Application Server: JBoss

The JBoss and Enhydra application servers appear to be the two predominant freely-available application servers. JBoss has integrated Tomcat into the application server itself, using Tomcat as the web container / web interface for the application server. Apache SOAP interoperates well with Tomcat, so JBoss is given the edge over Enhydra, which uses its own web server implementation.

Both application servers provide adequate database support for persistence, needed for satisfying requirement [SI-3: Persistent Context].

6.2 Messaging

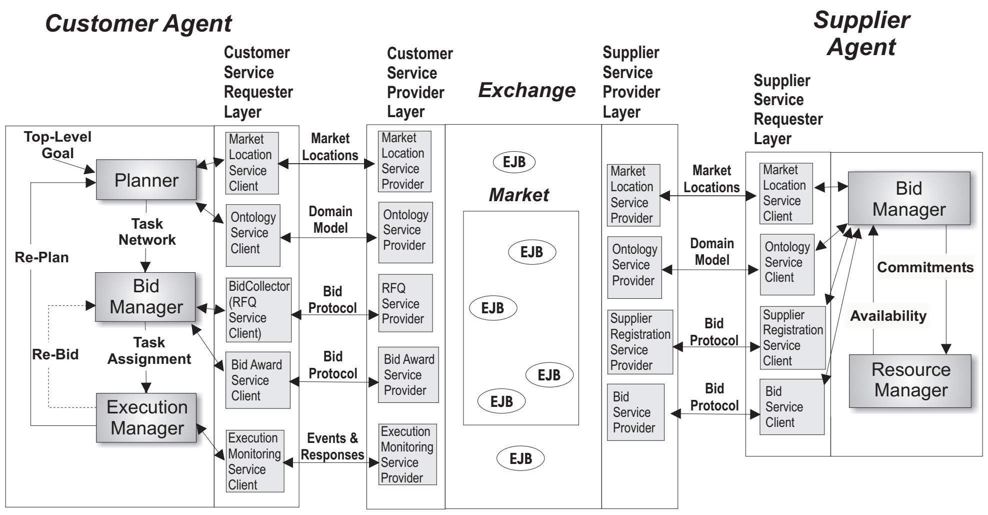
6.2.1 Messaging System: Apache SOAP

Apache SOAP also provides a means of messaging via a message router, SMTP, and POP3. The messaging system is free and Internet-standards based.

7 The Revised Magnet Architecture

7.1 Introduction

The revised MAGNET Architecture is similar to the original architecture in [7]. The components and interactions described in the reference model (see section 3) have not changed; the major changes in the architecture involve the move to a service-based model for communication and the distribution of MAGNET market components (exchange, market, and session) into a Enterprise JavaBeans-based architecture, as shown in Figure 5.



Customer Agent
Customer Service Requester Layer
Customer Service Provider Layer
Exchange
Supplier Service Provider Layer
Supplier Agent
Supplier Service Requester Layer
Top-Level Goal
Planner
Task Network
Re-Plan
Bid Manager
Re-Bid
Task Assignment
Execution Manager
Market Location Service Client
Ontology Service Client
BidCollector (RFQ Service Client)
Bid Award Service Client
Execution Monitoring Service Client
Market Locations
Domain Model
Bid Protocol
Bid Protocol
Events & Responses
Market Location Service Provider
Ontology Service Provider
RFQ Service Provider
Bid Award Service Provider
Execution Monitoring Service Provider
EJB
Market
EJB
EJB
EJB
EJB
EJB
Market Location Service Provider
Ontology Service Provider
Supplier Registration Service Provider
Bid Service Provider
Market Locations
Domain Model
Bid Protocol
Bid Protocol
Market Location Service Client
Ontology Service Client
Supplier Registration Service Client
Bid Service Client
Bid Manager
Commitments
Availability
Resource Manager

Figure 5: The Revised MAGNET Architecture

7.2 Customer Agent

The Customer Agent interacts with the Market in order to do planning, manage bids, and monitor execution of the tasks corresponding to the awarded bids. In the revised MAGNET architecture, the Customer Agent interacts with the market as a SOAP service requester, preferably through the use of the requester API. The API contains separate classes that act as SOAP service requesters for each of the SOAP services available to customer agents in the MAGNET system. These services include:

- A market location service for finding available markets in a MAGNET exchange.
- An ontology service for obtaining the market domain model and statistics from a market.
- A Request For Quotation (RFQ) service for submitting a RFQ into a market and establishing a means of bid collection for the customer agent.

- A Bid Award service for notifying suppliers in a market of awarded bids.
- An execution monitoring service (or services) for providing a means to monitor the execution of a task or tasks corresponding to awarded bids (execution monitoring in MAGNET is currently an open issue).

7.3 Supplier Agent

The Supplier Agent interacts with the Market in order to obtain information about markets and to bid on tasks or sets of tasks based on the RFQs in the market. Like customer agents, supplier agents use a service requester API for ease of composing SOAP requests. The Supplier API contains separate classes that act as SOAP service requesters for each of the SOAP services available to supplier agents in the MAGNET system. The services are:

- A market location service for finding available markets in a MAGNET exchange.
- An ontology service for obtaining the market domain model and statistics from a market.
- A supplier registration service for registering the supplier's intent to bid on selected types of tasks in a particular market.
- A bid service for submitting bids on tasks in an RFQ.

7.4 Exchange

The Exchange exists as a set of Enterprise JavaBeans (EJBs) in the MAGNET environment, allowing the application server to distribute exchange functions as necessary across the enterprise. Currently, the ability to locate markets is the only exchange-related function supported by the MAGNET system. As stated in Sections 7.2 and 7.3, market location capabilities are exposed to Customer and Supplier Agents by means of a SOAP service provider in the MAGNET system. The market location SOAP service in the MAGNET system calls methods on a proxy EJB that interacts with the rest of the MAGNET system. The proxy EJB allows the SOAP service provider implementation to be as simple as possible and to be isolated from the exchange implementation itself. See [10] for further discussion of the advantages of the proxy design pattern.

Note that many other approaches exist for agent market location. See Section 9.2.1 for a discussion of another approach that merits further investigation.

7.5 Market

The concept of MAGNET Markets also exists within the MAGNET system as EJBs.

Customer agents interact with the EJBs representing MAGNET Markets through the use of SOAP services for obtaining Ontology information (currently, only statistical information about the type of tasks a market supports), submitting RFQs, and awarding bids. An execution monitoring-related SOAP service provider is also expected to exist in the MAGNET system in the future. The SOAP service providers interact with the MAGNET system through the use of a proxy EJB.

Supplier agents also interact with the EJBs representing MAGNET Markets through the use of the MAGNET system's SOAP service providers. As noted in Section 7.3, Supplier Agents may use the SOAP services to obtain Ontology information, for registration in Markets, and to submit bids against tasks in submitted RFQs. Supplier-related SOAP service providers access the MAGNET system through a different proxy EJB than customer-related SOAP service providers (thereby allowing the types of agents to be treated differently in the proxy object layer if desired).

7.6 Market Sessions

Market Sessions exist within the MAGNET system as EJBs and are created and/or accessed through use of the market-related SOAP services described in Section 7.5. Session persistence is addressed through the use of entity beans (which are persisted as needed through the use of the application server's database). Session-related EJBs are used by the RFQ, bid submission, and bid award SOAP services.

8 Prototype Implementation Work

8.1 Introduction

The revised MAGNET Architecture described in Section 7 of this paper has been partially implemented as part of the work done for this Plan B Project. The intent was to prove the validity of the new architecture by showing that the recommended technologies work for implementing the new architecture. As such, it is not a complete implementation of the new architecture and focuses on using Apache SOAP for Customer Agent to Market communications. The work described in this section was also done to get the MAGNET team started on implementing the new architecture.

8.2 Apache SOAP Integrated Into Communications Framework

Agent-Market communications exist as SOAP service requests and responses in the new MAGNET architecture. The existing implementation for Customer Agent-Market communications (Java RMI) was replaced with Apache SOAP as part of this Plan B Project.

8.2.1 Entire Protocol SOAP-serializable

One issue with the SOAP data encoding scheme is that the SOAP specification only describes the mapping of simple data types to and from XML; SOAP intentionally leaves serialization of complex objects up to SOAP service requesters and providers. One of the nicer features of Apache SOAP is that Apache SOAP provides a robust Java API, including several extra classes for serialization and deserialization of Java objects, including the Java Date class and any class that follows the JavaBean conventions. This allowed the reuse of all the existing MAGNET protocol and customer protocol classes with a minimum of effort.

In general, most MAGNET protocol classes follow the JavaBeans convention which allows Apache SOAP to serialize them with no additional effort. Difficulty arose with the protocol classes that had complex state that could not be completely read and written through JavaBean-standard assessor and mutator methods (the Bid and RFQ classes). The current workaround for the problem is a set of custom classes (MagnetXMLReader and MagnetXML-

Writer) that can encode and decode these two protocol classes to XML, which is then put into a SOAP message as a simple String parameter. This is not an ideal means of serialization and deserialization and is mentioned as an area for future work in Section 9.2.2.

8.2.2 Example: SOAP-encoded JobInfo request

This section contains an example of the SOAP-encoded response from the JobService (the Ontology SOAP Service Provider) to a request for ontology information from a Customer Agent (the SOAP service requester). The response is a SOAP message containing an array of three JobInfo objects (only one is shown in the XML for brevity). The JobInfo class is a simple MAGNET class containing information about a particular task type supported by a MAGNET market. (In the XML below, whitespace has been added for clarity):

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:getJobInfosResponse
      xmlns:ns1="urn:JobService"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <return
      xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
      xsi:type="ns2:Array"
      ns2:arrayType="ns1:edu.umn.magnet.protocol.JobInfo[3]">
      <item xsi:type="ns1:edu.umn.magnet.protocol.JobInfo">
        <expectedDuration xsi:type="xsd:double">
          11.0
        </expectedDuration>
        <supplierAvailability xsi:type="xsd:double">
          0.11
        </supplierAvailability>
        <frequency xsi:type="xsd:double">1.0</frequency>
        <taskType xsi:type="xsd:string">Task Type 1</taskType>
        <description xsi:type="xsd:string">
          Dave\&apos;s neato JobInfo No. 1
        </description>
        <priceSigma xsi:type="xsd:double">
          0.11
        </priceSigma>
        <resourceAvailability xsi:type="xsd:double">
          0.11
        </resourceAvailability>
        <durationSigma xsi:type="xsd:double">
          0.11
        </durationSigma>
        <expectedPrice xsi:type="xsd:double">
          1100.0
        </expectedPrice>
      </item>
      ...
    </return>
    </ns1:getJobInfosResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

8.3 MAGNET Customer Agent Integration

In the new MAGNET architecture, the Customer Agent interacts with the market as a SOAP service requester through the use of the requester API, which consists of separate classes that act as SOAP service requesters for each of the SOAP services available to customer agents in the MAGNET system. The following table lists the Service Requester class for each service in the architecture. The table also lists the service provider used by the customer agent in the MAGNET server.

Service (Architecture)	Service Provider (Implementation)	Service Requester Class (Implementation)
Market Location Service	MarketListingService	MarketListingServiceClient
Ontology Service	JobService	JobServiceClient
RFQ Service (Synchronous)	SynchRFQService	SOAPSynchBidCollector
RFQ Service (Asynchronous)	(Not Implemented)	(Not Implemented)
Bid Award Service	(Not Implemented)	(Not Implemented)
Execution Monitoring Service	(Not Implemented)	(Not Implemented)

The MarketListingServiceClient and JobServiceClient classes are used by the Customer Agent once, at the time of startup (via method calls in XAuto-Customer and MagnetClientMain) to access the available markets and task types.

The SOAPSynchBidCollector is a synchronous service requester class; it will post an RFQ to the RFQ service and wait for all the resulting bids to come back from the market. The synchronous Bid Collector is simple and good for testing. It is expected that an asynchronous service will be implemented as well, where the Customer Agent posts an RFQ into the MAGNET system along with contact information; the system then responds with Bids from the market as they are submitted from suppliers. Bid Collectors are used by the Bid Manager component in the Customer Agent; the type of Bid Collector to be used can be configured at runtime through the use of the Bid Collector Factory and a property corresponding to the Bid Collector class to be used.

Bid awards and Execution Monitoring are not currently supported by the MAGNET system, so these services (requesters and providers) are not yet implemented.

8.4 MAGNET Server Integration

The SOAP service provider classes used by the MAGNET system to expose services to agents interact with a proxy EJB on the MAGNET server. The following table summarizes the implemented SOAP service providers and the method(s) on the proxy EJB (the CustomerSession session bean) that are used to handle the request and return results:

Service (Architecture)	Service Provider (Implementation)	CustomerSession Method (Implementation)
Market Location Service	MarketListingService	getMarkets()

Ontology Service	JobService	setMarketName()
		getJobInfos()

RFQ Service (Synchronous)	SynchRFQService	setMarketName()
		getOpenSessions()
		createNewSession()
		submitRFQInSession()
		getBidsForRFQInSession()

RFQ Service (Asynchronous)	(Not Implemented)	(Not Implemented)
Bid Award Service	(Not Implemented)	(Not Implemented)
Execution Monitoring Service	(Not Implemented)	(Not Implemented)

* As Supplier Agent integration has not taken place yet, it is not possible for the getBidsForRFQInSession() method to return any bids. The SOAP service currently generates its own bids in response to an RFQ and returns them to the Customer Agent for testing purposes.

9 Conclusions and Future Work

9.1 Conclusions

9.1.1 Importance of Architecture

Performing an analysis of the MAGNET architecture before design and implementation could have saved the MAGNET team time and effort. In particular, requirements analysis and thought about MAGNET stakeholders (see Section 2.1) would have led to discovery and documentation of the need for MAGNET to service distributed agents over the Internet (the [BAP-1: Open Communications] requirement). A lot of effort was put into implementing MAGNET communication protocols that later had to be discarded. The original MAGNET system was based on Java RMI, which is a “binary” protocol and does not satisfy requirement [BAP-1: Open Communications]. CORBA was analyzed and suggested for MAGNET implementation as well [7]. An Internet standards-based approach (such as using SOAP or combining XML and HTTP) might have been tried sooner if requirements had been better understood and documented.

Documentation of architecture requirements and reference models helps to avoid “Architecture By Implication” [2] since the architecture can be measured against the needs of stakeholders. Documentation of the architecture derived from the requirements allows design and implementation to take place within a high level structure, giving developers a common view of the system to be built. A goal of this Plan B Project has been to place structure on further MAGNET development through architecture design; the new architectural framework clearly illustrates that extending MAGNET is just a matter of implementing another SOAP service!

9.1.2 Internet-based Standards Powerful for Agent Communication

An important theme emerged during the development of this paper: the Breadth of Agent participation is crucial to the success of MAGNET and the key to providing the most opportunity for agent participation is to base MAGNET on practical, widely used Internet standards. Furthermore, a standards-based approach should be used in both the network transport and data encoding portions of MAGNET communications; it is not enough to support one or the other. Non-standards based transports do not traverse network boundaries well (for example, consider HTTP versus Java RMI.) Non-standards based data encoding leads to increased complexity in agent implementations and rigid constraints on the content of messages; the agent must understand the encoding and the message itself.

Apache SOAP was built on top of popular, accepted Internet standards like HTTP and XML. The power of this approach became obvious with the beginning of the prototype implementation; changes could be made on agent code at home and tested against a market running on a server at the University of Minnesota! The move to a standards-based approach meant that network boundaries were of lesser concern; the agent communication took place through the Internet Service Provider and University firewalls.

The development and use of MAGNET agents has become open to a much larger community. Agents are free to negotiate within a MAGNET environment which now may stretch across the entire Internet. Let the games begin!

9.2 Future Work

This section briefly discusses future work related to the work performed for this paper.

9.2.1 Exchanges and Service Discovery

A problem with the new MAGNET architecture is that of the non-standard means by which services are discovered.

The ability to locate markets is an exchange-related function supported by the MAGNET system. Market location capabilities are exposed to Customer and Supplier Agents by means of a SOAP service provider in the MAGNET system. The market names are merely listed by the service if the agent has access to them. However, there is no standard way for agents to first find the market location service, to find the exchange, or to find any other SOAP service supported by the MAGNET system (currently the MAGNET system is specified by an URL in the agent's property file). Two interesting technologies that may be used to overcome these limitations are UDDI and WSDL.

Universal Description, Discovery, and Integration (UDDI) UDDI is a standard for dynamic lookup, binding, and publishing of SOAP services. It allows to queries of different UDDI registries to look up businesses, information by business category, and service information. Agents could use UDDI to find MAGNET services; UDDI should be investigated further. See `http://www.uddi.org` for details.

Web Services Description Language (WSDL) A WSDL file may be used to define a web services based on SOAP. WSDL files are being adopted on both the client and server portion of SOAP communication to better describe the type of communication that will occur between the two parties.

For instance, a WSDL file may describe that a particular message is sent as input to a SOAP server and a particular message is sent in response to that input. A WSDL file also describes at which URL and port a particular Web service exists. MAGNET agents could access a UDDI directory to obtain a WSDL description of the web services supported. WSDL should be investigated as a means of publishing MAGNET services.

9.2.2 Implementation

The following implementation tasks need to be completed:

- Finish Customer SOAP services for bid awards and execution monitoring.
- Implement the Supplier Agents services (Market Location, Ontology, Supplier Registration, and Bid Submission).
- Asynchronous communications are not implemented; in particular, customers should be able to receive bids asynchronously by submitting RFQs to the correct service, and Supplier Agents should be able to receive RFQs asynchronously by registering in the Magnet system using the appropriate service.
- The Bid and RFQ protocol classes should follow the JavaBeans standards closely so that they may be easily serialized and deserialized using Apache SOAP; the current scheme of custom encoding classes requires distribution of a common Document Type Definition (DTD) and encoding classes that could be avoided entirely if standard SOAP serialization could be used.

- Security should be implemented; an initial step would be to use Tomcat's SSL (Secure Socket Layer) support for "wire" level security. Higher-level security (authentication and the like) can be achieved through the use of the SOAP envelope, which can contain multi-part (MIME-encoded) information for items such as public keys.

References

[1] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice.* Addison Wesley Longman, Inc., Reading, Mass., 1998.

[2] William J. Brown, Raphael C. Malveau, Hays W. McCormick III, and Thomas J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis.* John Wiley & Sons, Inc., New York, NY, 1998.

[3] Anthony Chavez and Pattie Maes. Kasbah: An agent marketplace for buying and selling goods. In *Proc. of the First Int'l Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK, April 1996.

[4] John Collins, Corey Bilot, Maria Gini, and Bamshad Mobasher. Mixed-initiative decision support in agent-based automated contracting. In *Proc. of the Fourth Int'l Conf. on Autonomous Agents*, pages 247–254, June 2000.

[5] John Collins, Rashmi Sundareswara, Maksim Tsvetovat, Maria Gini, and Bamshad Mobasher. Multi-agent contracting for supply-chain management. Technical Report 00-010, University of Minnesota, Department of Computer Science and Engineering, Minneapolis, Minnesota, 2000.

[6] John Collins, Maxim Tsvetovat, Bamshad Mobasher, and Maria Gini. Magnet: A multi-agent contracting system for plan execution. In *Proc. of SIGMAN*, pages 63–68. AAAI Press, August 1998.

[7] John Collins, Ben Youngdahl, Scott Jamison, Bamshad Mobasher, and Maria Gini. A market architecture for multi-agent contracting. In *Proc. of the Second Int'l Conf. on Autonomous Agents*, pages 285–292, May 1998.

[8] Peyman Faratin, Carles Sierra, and Nick R. Jennings. Negotiation decision functions for autonomous agents. *Int. Journal of Robotics and Autonomous Systems*, 24(3-4):159–182, 1997.

[9] David Flanagan, Jim Farley, William Crawford, and Kris Magnusson. *Java Enterprise in a Nutshell.* O'Reilly & Associates, Inc., Sebastopol, CA., 1999.

[10] Erich Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley Longman, Inc., Reading, Mass., 1995.

[11] Dan Gisolfi. Web services architect, part 1: An introduction to dynamic e-business. Technical report, IBM, 2001.

[12] Dan Gisolfi. Web services architect, part 3: Is web services the reincarnation of CORBA? Technical report, IBM, 2001.

[13] Object Management Group. The common object request broker: Architecture and specification. Technical report, Object Management Group, Inc., 2001.

[14] Jason Hunter and William Crawford. *Java Servlet Programming.* O'Reilly & Associates, Inc., Sebastopol, CA., 1998.

[15] Tarak Modi. Clean up your wire protocol with soap, part 1. *JavaWorld*, March 2001.

[16] Robert Orfali, Dan Harkley, and Jeri Edwards. *Instant CORBA.* John Wiley & Sons, Inc., New York, NY., 1997.

[17] Dirk Reinshagen. XML messaging, part 1. *JavaWorld*, March 2001.

[18] Tuomas W. Sandholm. *Negotiation Among Self-Interested Computationally Limited Agents.* PhD thesis, Department of Computer Science, University of Massachusetts at Amherst, 1996.

[19] Bill Shannon. Java 2 platform enterprise edition specification, version 1.3. Technical report, Sun Microsystems, Inc., 1999.

[20] Katia Sycara and Anandeep S. Pannu. The RETSINA multiagent system: towards integrating planning, execution, and information gathering. In *Proc. of the Second Int'l Conf. on Autonomous Agents*, pages 350–351, 1998.

[21] Michael P. Wellman and Peter R. Wurman. Market-aware agents for a multiagent world. *Robotics and Autonomous Systems*, 24:115–125, 1998.

[22] Peter R. Wurman, Michael P. Wellman, and William E. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In *Second Int'l Conf. on Autonomous Agents*, pages 301–308, May 1998.

[23] Edward Yourdon. *Software Reusability:The Decline and Fall of the American Programmer.* Prentice-Hall, Englewood Cliffs, NJ, 1993.