Metabroker: A Generic Broker for Electronic Commerce

Steve Caughey, David Ingham & Paul Watson

Department of Computing Science, Newcastle University,
Newcastle upon Tyne, NE1 7RU, UK.

{s.j.caughey | dave.ingham | paul.watson}@ncl.ac.uk

Abstract

In traditional commerce, brokers act as middlemen between customers and providers, aggregating, repackaging and adding value to products, services or information. In the broadest sense of the term newspapers, travel agents and department stores can all be thought of as performing a brokering role. In today's Web, such services are lacking, with the result that individuals are forced to manually discover, collate and analyse information to meets their needs.

In this paper, we highlight the requirement for brokering services for electronic commerce and describe the design of the Metabroker system, a generic framework for creating electronic brokers. The aim is to provide a framework that provides commonly-required functionality and support for popular communication protocols and data formats. Specialist brokers are then created by populating the base framework with the necessary business logic to support the area of speciality of the broker. Our design integrates distributed object, metadata and object database technologies.

Keywords: electronic commerce; brokering; metadata; distributed objects

Introduction

From a commercial perspective, the Web has promised much more than it has delivered and the dream of world-wide Internet business enterprises has yet to become a reality. A number of businesses are making their services available on the Internet. However, electronic commerce currently resembles a vast, sprawling bazaar in which visitors must wander through countless market stalls, browsing casually, pausing at a particularly attractive stall before moving on. For most potential customers, this is frustrating and time-consuming.

In traditional commerce, middlemen, or brokers, make it easier for customers to find, compare and buy because they aggregate goods and services from a variety of sources and display them in a way which is helpful to customers. Our definition of a broker is perhaps more general than most; we consider department stores, home shopping catalogues, newspapers and travel agents to be all serving a brokering role in that at some level the service presented to their clients is generated through the use of information, services or products provided by others.

A number of services on the web today provide primitive brokering functionality; for example, CDnow sells the products of many CD and video suppliers and also provides independent product reviews and in-store accounts. Another example is BarclaySquare which provides a shopping mall abstraction that houses a number of independent service providers. The mall provides value-added features including a uniform shopping cart and banking system together with centralised management of user details and preferences. However, electronic brokering offers the potential for vastly more sophisticated services; examples of how brokers can add value include:

- aggregating the services offered by several providers and presenting them in a consistent format to customers. A Web version of a home shopping catalogue could do this by combining product information from services provided by a set of manufacturers.
- querying a set of services in order to meet a client's specific requirements. For example a travel information broker with knowledge of a customer's preferences could extract suitable holidays from the information provided by a set of travel companies.
- monitoring a set of services and informing clients of anything new which may interest them. For example a travel information broker could inform clients of cancellations to flights on which they were booked.
- providing local services in the broker which are in addition to, but build on, those offered by a set of providers. For example, a shopping broker which offers for sale the goods produced by several manufacturers may offer a service whereby clients can be alerted when an out-of-stock article becomes available again, even if the manufacturers themselves do not offer such a service.
- combining information from several services. For example a travel information broker may offer timetable information for journeys which require multiple forms of transport (e.g. journeys which require taking a train to an airport in order to catch a flight). To achieve this, the broker will have to combine timetable information from a set of services provided by different travel service providers.

In the electronic marketplace of the near future, there are likely to be large numbers and varieties of brokers. We believe that the widespread and rapid deployment of brokering services depends upon the availability of building blocks which are sufficiently generic that they can be tailored to produce a wide range of specific brokers. We call this collection of building blocks a **generic broker**. If such a generic broker framework is not available then specialised brokers will be implemented in an ad-hoc manner with a consequent wastage of effort, while their construction will be beyond the resources and skills of many companies who would otherwise benefit from them.

In this paper we present the design of the *Metabroker* system, a generic framework for the construction of specialist electronic brokers. Our design is based upon the integration of distributed object, metadata and object database technologies. The remainder of the paper begins with an analysis of the generic requirements of an electronic broker from the perspectives of clients, service providers and broker administrators. The next section provides a description of the Metabroker design, illustrating how the aforementioned requirements are met. We then describe our proposed implementation plans including descriptions of the experimental prototypes that have been used as a test-bed for the ideas. Finally we draw conclusions from our work, compare it with related work, and point to future directions.

Generic electronic brokering requirements

An electronic broker utilises a set of resources supplied by external service providers in order to provide value-added services to clients. A high level view of the external interactions of an electronic broker is shown in Figure 1 below. One of the key requirements of such brokers is the ability to accommodate multiple ways of interacting with clients and resources. In this section, we shall consider the requirements from the perspective of clients, resources and also brokering service providers.

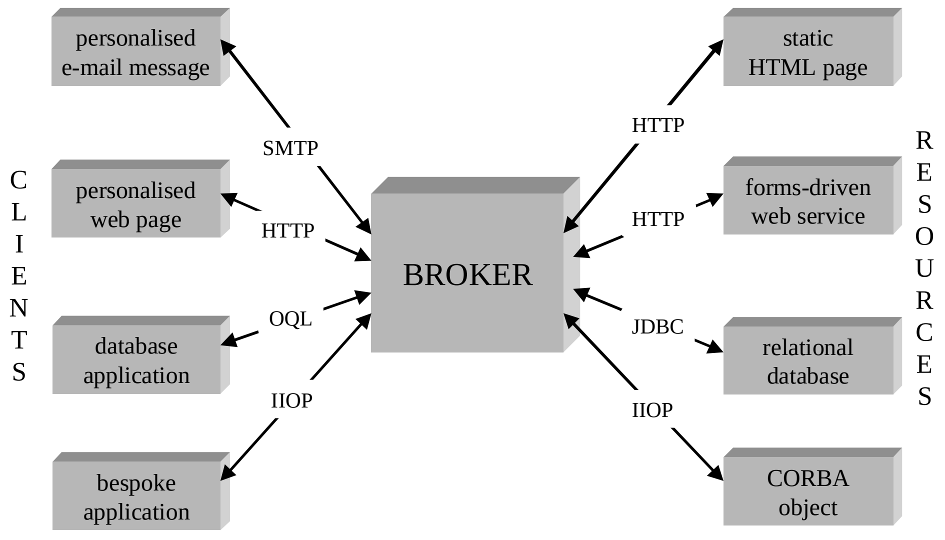

Figure 1: External broker interfaces

Client requirements

From the clients' perspective, a broker is required to serve as a *one-stop-shop* for services, products, or information in a specific field. Clients therefore benefit by not having to trawl through a myriad of autonomous service providers, collating and analysing information manually. A travel agent broker, for example, may provide a journey planning service that utilises timetable and availability services from numerous airlines, rail and bus companies. Additionally, the broker may provide accommodation booking facilities utilising services provided by hotel chains. Added value services such as travel guides, currency conversion or weather reports may also be provided.

In general, clients would like services to adapt to their needs rather than requiring them to adapt to the capabilities of the service. One aspect of this adaptability is the support for a range of ways of interacting with the broker. Due to the ubiquity of Web browsers, it is likely that HTTP will be the primary interface, however other possibilities include e-mail, CORBA, e-mail, pagers, OQL and newsgroups.

Another potential benefit for clients of an electronic broker is the ability to personalise the interaction that a particular client experiences with the broker. Aspects of personalisation could include content delivery; for example, a client could choose whether to have results delivered as an email message or rendered as a personalised Web page. Additionally, the broker could maintain information about the client regarding particular preferences, both broker-specific, such as their closest airport, or more general information such as liking for HTML frames.

A particular broker will likely offer a range of services to clients varying in their complexity. Some classes of request may be instantly satisfied by the broker and therefore could be offered as a synchronous operation perhaps through a forms-based Web service. Conversely, many requests are likely to involve considerable data gathering and analysis, therefore necessitating an asynchronous mode of operation. Additionally, a broker could support an mode of operation in which it is continuously monitoring for a specific set of conditions on behalf of a client. For example, a client of a travel agent broker may request to be informed of any special deals for holidays to the Greek Islands. This mode of interaction is often referred to as *agent-based* with the needs of an individual being represented by an autonomous agent executing within the broker (more advanced forms of agent-based interaction are considered in later sections).

Resource requirements

The success of the broker will be measured by its ability to provide useful service to clients and this ability will be heavily influenced by the volume and quality of resources to which it has access. Even in a specific field, it is unlikely that service providers will agree on standard ways to present their resources. For instance, despite the fact that the majority of service providers on the Internet today use HTTP as the protocol of choice, with HTML as the preferred content format, there is little commonality in how information is presented. It would be impractical to require service providers to repackage their resources to meet the needs of the broker. Instead the broker has to be extremely flexible in its ability to interact with a wide range of protocols and data presentation formats.

As an example, consider one of the airline timetable services available on the Web today. Clients interact with the service using an HTML form that requests information including origin, destination and the preferred dates of travel. The service responds with an HTML page containing a tabulated list of possible flights with each entry comprising flight number, departure time, arrival time, type of plane etc. In order to exploit such a service in our example travel agent broker, the broker must be able to drive the HTTP protocol, passing the necessary input data to the service, and then be able to extract the salient information from the HTML that is returned. This is an example of a service that is providing no co-operation with the broker and so the broker is required to operate with the existing interfaces of the service. However, the degree of co-operation provided by services is likely to vary. Co-operation is likely to increase as a broker becomes more popular since it is in the interest of the service provider to make its resources available to the broker's client base. To this end, service providers may provide access to the raw data that is used by the Web service front-end. Such data may be held in a database or a CORBA object.

Other forms of co-operation may exist, for example, a service provider may choose to inform the broker when resources are updated. In general, to support these levels of co-operation interfaces are required that allow the service provider to tailor the broker's interaction with resources. In the extreme case, a service provider may choose to provide only the raw data, outsourcing the provision of services completely to the broker.

Service provider requirements

In addition to the functional requirements of clients and resources, there exist a number of requirements imposed by the provider of the broker service, including performance, management and revenue gathering.

The performance required by different broker providers will depend upon i) the rate of client access, ii) the amount of computation required within the broker and iii) the amount of communication required with resources. The first and second of these dependencies can be satisfied by the choice of a suitable hardware platform with the required networking and computational abilities. A truly generic broker therefore should be capable of executing on a large number of different platforms. A broker which is highly dependent upon communication with resources is vulnerable to both performance loss and/or failure due to network congestion or resource unavailability. However this dependency may be decreased through the use of resource caching. The choice of caching policy and of the relevant resource data which requires caching is entirely dependent upon the use of individual resources within the broker. In many cases caching some set of data about the resource, rather than the entire resource, will be sufficient to satisfy many client accesses. A broker should therefore be capable of flexible caching policies which can be applied to individual resources as required.

A broker is a complex piece of software which requires management tools to allow it to be configured and monitored during its execution. If it is to be capable of evolution then it must be possible to reconfigure the broker to satisfy changing requirements. In particular the broker should be capable of utilising additional hardware resources to cater for growth. It should also be possible to introduce new functionality into the broker and to take advantage of new protocols and data formats as they are deployed within the Web. Ideally all reconfiguration should occur without a loss of service and with the minimum of intervention.

Brokers can obtain revenue from some combination of four possible sources: i) from clients, ii) from resource providers, iii) from advertising and iv) by providing a useful service funded by an external organisation. A generic broker should therefore provide: the means to monitor usage for the purposes of charging clients and resource providers; the ability to offer personalised advertising; and, some means by which to collect revenue.

Metabroker Design

We will first present the overall architecture of our generic broker, which we call *Metabroker*. Metabroker comes in two parts, a Generic Broker Module and a Specific Broker Module. The *Generic Broker Module* (GBM) provides a distributed object framework and a set of commonly useful services. This module also contains a run-time library of the common data types (and protocols) encountered on the Internet. All objects within the GBM offer a common set of useful interfaces and are either instances of the data types offered by the GBM or by the Specific Broker Module. The *Specific Broker Module* (SBM) configures and drives the GBM, supplying the functionality and data types which represent the specific intelligence of the broker i.e. the ability to extract information from the data and to react sensibly to the information obtained.

We begin with a description of the GBM. As we have already described, the external entities of concern to a broker are the clients who wish to make use of the broker and the resources which the broker deems of interest to clients. Within the Metabroker, proxy objects (or just *proxies*) are used to represent and maintain information relating to clients and resources. In addition to these external entities the Metabroker will also have its own internal entities. These will include services offered by the broker, services used internally, and entities used to record complex relationships between entities within the broker. All of these entities are also structured as objects, which we collectively call *relationship objects*. Objects may hold references to, and thereby invoke operations upon, other objects.

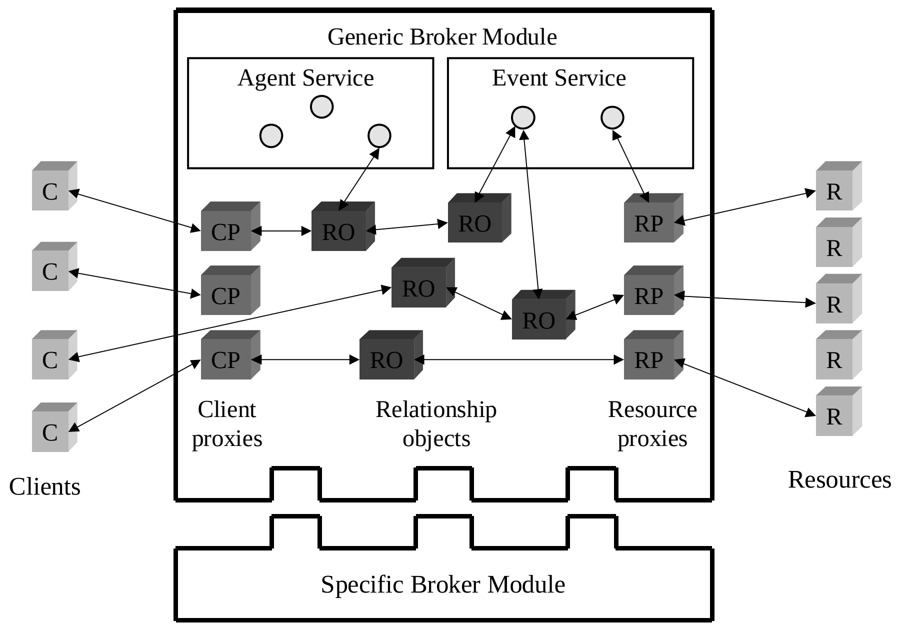

Figure 2: Metabroker architecture

Objects can register their interest in specific events with an *Event Service*, or can inform the service of events they can generate and of objects to be informed when this occurs. Typical events include client actions, updates to resources (recognised by their proxy) the introduction or removal of proxies, and timing events.

An *Agent Service* allows clients and resource providers to create agents which can respond on their behalf to the occurrence of specific events. Agents execute within a secure environment i.e. without causing malicious or accidental damage to the Metabroker. Frequently the event and agent services will interact; for example, a client might create an agent to look for new resources containing some set of interesting keywords and have the agent inform the client by e-mail when it succeeds. The agent can register its interest in the creation of new resource proxies with the event service, and when a new proxy is created the agent is signalled and examines the proxy to see if it contains the necessary keywords. Both agents and events are implemented as objects.

That part of the broker which is specific to some particular information domain is envisaged as a plug-in Specific Broker Module (SBM). This configures the GBM, populating it with the necessary objects and then monitoring its behaviour. It is capable of on-line reconfiguration where this is deemed necessary. The objects created by the SBM are:

- proxies for the initial set of resources;
- relationship objects which represent useful services (either for internal or client use);
- a set of events to be signalled and
- a set of agents.

Once configured the Metabroker can go live, listening for and responding to clients.

In the following sections we will examine each of the components of the Generic Broker Module in more detail i.e. proxies, relationship objects, the event service and the agent service. But first, as every entity within the broker is structured as an object, we begin by discussing the generic functionality of all Metabroker objects.

Broker Objects

Every entity within a Metabroker is structured as an object. An object encapsulates data and provides one or more interfaces. Each interface is accessible using one or more protocols. Requesting a service from some object is termed an invocation upon that object and is achieved by passing a message to the object. All Metabroker objects have Metabroker internal interfaces accessible within the Metabroker, but objects are also capable of allowing external access to some of their interfaces via one or more protocols including http, RMI, IIOP etc.

At a minimum every object supports the following three Metabroker internal interfaces :

- A caching interface. Each object is capable of maintaining a cache of all or part of the data returned in response to earlier accesses upon other entities. An object maintaining such a cache may therefore be capable of satisfying requests which might otherwise have required accesses upon other entities. This use of caching may be especially important when it eliminates an access to a remote resource. In this case it can decrease latency, decrease communication costs, and improve availability i.e. even if the entities to be accessed are unobtainable the cache might still be utilised. Of course caching does introduce the possibility of serving out-of-date information and so an object's caching interface allows flexible control over the data to be cached and over the consistency of that data.
- A content-metadata interface. *Content-metadata* is data **about** an object rather than data contained **within** it and is used to enable efficient searching. For example, imagine an object 'cubism4.gif' which is a picture of a painting by Picasso. In this case 'a painting by Picasso' is metadata, i.e., data pertaining to the object but not (necessarily) encoded within it. Support for content-metadata allows relationships to be recorded between objects, or between objects and abstract concepts, and is of great benefit when searching. Via this interface objects will allow content-metadata to be recorded in a format resembling that specified by Dublin Core, and served [Dublin]. Content-metadata can be generated for any object by interpreting the data returned by the object or by recognition of significant access patterns e.g. recognition that the object is accessed in conjunction with other objects who share some common metadata. Each item of content-metadata has an associated 'level of trust' indicator, to which values may be assigned.
- A management interface is used to manage objects within the broker. The interface supports referential integrity of broker objects i.e. objects which are referenced from elsewhere within the broker continue to exist, whilst objects with no references are automatically removed. Type-specific concurrency control is also provided, as are persistence and crash-recovery mechanisms and the ability to move objects within the broker should the broker span a number of machines.

Proxy objects

As we have already described there are proxies for two types of external entity - clients and resources, and each proxy encapsulates the entity for the Metabroker. The two types are in fact very similar in function with the distinction between them being largely indicative of their typical interactions with the broker i.e., clients tend to approach the broker for service, whilst the broker approaches resources.

The content metadata of a proxy object which represents a client may include information which is generic in the sense that it is independent of the nature of the broker (such as a client's name) as well as broker specific information (such as a client's local airport in the case of a travel agent broker). Likewise, content metadata for a resource proxy may be generic (such as the time of last modification) or broker-specific (such as the set of destinations served by a particular holiday company).

In addition to content metadata, every proxy object contains *access metadata*, i.e., data about how the external entity may be accessed. The access metadata held by our proxy objects includes the protocol used to communicate with the entity, i.e., a reference to the appropriate protocol handler object (e.g., an SMTP or HTTP protocol handler) together with data that is used to configure the protocol handler for the particular entities (e.g., a client's email address or a resource's URL).

In some circumstances, a broker may interact with a particular external entity using a number of protocols. In such cases the metadata stored by the proxy will be categorised by protocol. Access metadata is implicitly protocol specific whereas some items of content metadata will be protocol independent while others will be tied closely with a particular protocol. For example, data describing a client's preferred frequency of email delivery will be tied to the email protocol handler, whereas a client's local airport is generic information independent of the access mechanism used by the client. Figure 3 illustrates the contents of an example client proxy object.

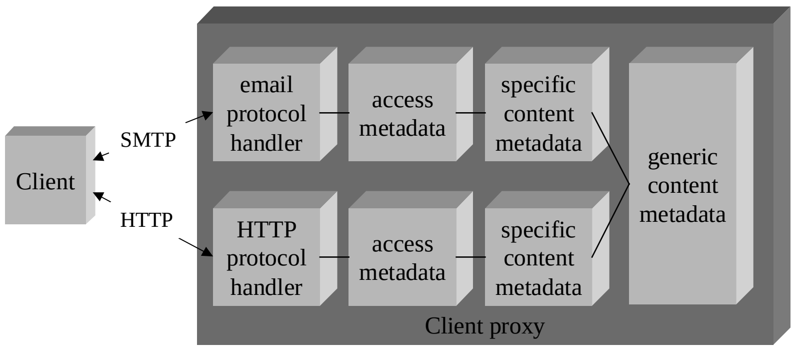

Figure 3: Example of a client proxy object

One of the important features of our design is the separation of delivery, content and presentation. Encapsulating the details of a particular protocol within a protocol handler eases the introduction of, or changes to, protocols used by external entities whether clients or resources. The separation of presentation from content is essential if different clients wish to receive the same information via different delivery mechanisms, e.g., a pager message has to be restricted to the concise facts whereas an email message could be more detailed. Additionally, this enables us to tailor content based upon the preferences of an individual client, e.g., a client's liking for the use of HTML frames. At the resource side, this separation helps to isolates the broker (and therefore the broker's clients) from changes in the native presentation of resources. For example, if the layout of an HTML page changes then only the proxy representing that page is required to change.

Relationship Objects

It is part of the function of the Metabroker to identify and present relevant relationships between clients and resource data. Relationship objects assist in this task by representing commonality between objects. A relationship object for example might contain all current references to proxies for whom the value for the content-metadata fields 'Picture' and 'Artist' are ‘true’ and ‘Picasso’ respectively. This type of object is acting as a cache of the results of a search across all proxies. Other relationship objects might involve related service interfaces. For example such an object might hold references to a number of resource proxies, each of which represents a service by which a particular airline’s flight details are obtainable but where the individual interfaces are all different. The relationship object might hide those differences by offering a single generic flight details interface whereby a single invocation amalgamates the results of access on all of the airline services. Particularly useful interfaces, such as this, can be published so as to be accessible from outside the broker i.e. as value-added services offered by the broker. As objects have the ability to present themselves as Web pages, clients can browse though the broker following links between proxies and relationship objects. The presentation (if any) of each object can be tailored to the client.

The Event Service

The event service is the means by which specific events can be registered and, when triggered, signalled to interested parties. These events include changes to a particular content-metadata field within an object, timing events and the introduction or removal of objects. Objects can register their interest in a particular event and if that event is triggered those objects are informed and can take appropriate action. For example, let us imagine a resource which represents a clothes catalogue with a content-metadata field ‘SEASON’. Clients could register their interest in changes to the value of this field so that whenever the value changed, all the proxies of interested clients would be signalled and could then inform their clients (perhaps through updates to the client’s personalised Web page or by sending e-mail).

The Agent Service

The Agent Service allows the introduction of non-trusted functionality into the broker. Agents are encapsulated within objects and with two notable exceptions are identical to other broker objects. These exceptions concern security restrictions. The first is that agents have a limited access to system primitives e.g. they cannot manipulate the underlying file system or fork new processes. The second is that all agent messages to other objects are automatically tagged with the identity of the agent. This enables other objects to restrict the access of all, or some, agents to all, or some of their services.

Clients of the broker can use the Agent Service to introduce agents which autonomously monitor the broker for the occurrence of specific conditions. These conditions include the existence of specific content-metadata fields with particular values and the recognition of specified information within data. The results of an agent’s work can be stored for later examination or passed to other objects. Agents are therefore capable of carrying out work within the broker which might otherwise have required complex and expensive communication between client and broker.

Resource providers can also introduce agents into the Metabroker which perform similar functions to client agents. These agents might be looking for changes introduced by their competitors or for particular client profiles to help in the targeting of their advertising.

Finally third-party service providers might want to introduce services which utilise the broker (as described earlier). These services can be introduced as agents.

Current status and future work

In order to validate some of the concepts presented in this paper, we have implemented several prototype systems. One of these prototypes was a simple toolkit for building brokers that utilise forms-based Web services. An example broker, known as the *Informed Traveller*, was built which provides a journey planning service encompassing both rail and air travel [Halsey97]. Clients interact with the service using a Java applet that communicates with the broker server using RMI. The broker utilises timetable services provided by Railtrack and British Airways. These services provide HTML forms-based interfaces to accept user data and provide their results in the form of tabulated HTML. The resource proxies were constructed using the Web Automation Toolkit from webMethods Inc. The toolkit can generate Java code for accessing forms-based HTML services, i.e., the protocol handler in our terminology. The access metadata comprises the address of the service together with Web Interface Definition Language (WIDL) data. WIDL is an application of Extensible Markup Language (XML) [Bray97, Khare97] and is used to specify how the service is driven and how the salient items of information are extracted from the HTML returned. The broker objects used caching to improve performance with query results being stored in a database so as to provide fast response for subsequent queries.

The aim of another prototype, *QObjects*, was to demonstrate how content metadata could be structured and queried in a generic manner [Shimshoni97]. Generic resource proxies for several data formats were created; these were then specialised to create proxies for specific classes of resource (e.g., Internet RFCs). Content metadata was extracted from the resources and stored within a POET object-oriented database. A uniform query interface, based upon attribute-value pairs was used to query the resources. Caching was used extensively to improve performance by reducing the need to access the remote resources.

We are currently in the process of creating the specification for a large scale broker application based upon a local community service which we hope to implement over the coming six months.

Our intention is to make the Java virtual machine [Lindholm96] the foundation for our architecture as it provides the necessary portability and support for online enhancement. We propose to use an object-oriented database management system to provide storage, query capability and transaction support for the large number of objects likely to be held within a Metabroker. The Shadows distributed object support system [Caughey93] will be used to provide the generic functionality, in particular, the flexible caching [Caughey97] and referential integrity mechanisms [Ingham96] required of broker objects. W3Objects facilitates in the construction of highly customisable proxies in which presentation and content are clearly separated [Ingham97]. We are investigating existing agent and event services for incorporation within Metabroker. The diagram, shown in Figure 4, gives a high-level overview of the proposed software architecture for Metabroker.

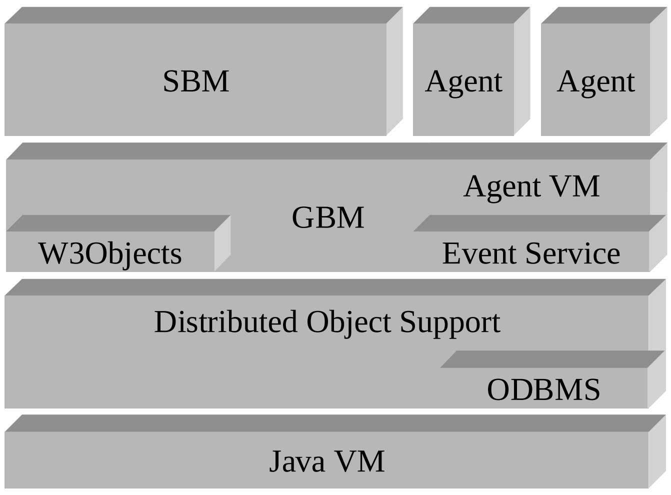

Figure 4: Metabroker software architecture

Conclusions

In common with other researchers [Chavez96, Fido], we believe that one of the necessary ingredients for the success of electronic commerce is the provision of brokering services similar to those that exist in traditional commerce. Our approach is similar to that adopted with the Smart Catalogs architecture [Keller95] in that we wish to retrieve information from multiple sources which we then present in a form suited to clients. However, we see the need for a generic brokering framework which enables the construction of specialist brokers, an example of which could be a smart catalogue.

We conclude that in order to be successful a broker has to be flexible enough to adapt to the needs of the clients and service providers rather than force them to conform to the broker's standards. This adaptation encompasses the support for a multitude of presentation formats and delivery protocols. However, we need to represent these diverse external entities as uniform internal entities which can be manipulated in a consistent fashion within the broker. We achieve this through the use of the polymorphism inherent in object-oriented technology. Our aims in this respect are similar to those of MetaMagic [Shklar97] which supports the creation of virtual Web sites that consist of metadata representations of external entities. This separation of Web presentation from content allows a MetaMagic server to offer multiple views on existing content. This technology corresponds to our use of resource proxies. However, we extend the model to include clients enabling us to make appropriate connections between the needs of clients and the services provided by resources, i.e., to support brokering.

The experience gained in implementing our prototypes has convinced us of the viability of the Metabroker project. We aim to utilise components from these prototypes together with our previous experience in distributed object services [Parrington95, Caughey93, Ingham95] to implement the Metabroker architecture.

Acknowledgements

The prototype systems described in this paper were implemented by Stephen Halsey and Daniel Shimshoni as part of their Masters degrees. Their work clarified some early design ideas and helped to verify the feasibility of the project.

References

[Bray97] T. Bray and S. DeRose, “Extensible Markup Language (XML): Part I. Syntax,” World Wide Web Consortium Working Draft (work in progress), November 1997.
<URL:http://www.w3.org/TR/WD-xml>

[Caughey93] S. J. Caughey et al., “SHADOWS: A Flexible Support System for Objects in a Distributed System,” Proceedings of the 3rd IEEE International Workshop on Object Orientation in Operating Systems (IWOOOS), Ashville, North Carolina, USA, December 1993.
<URL:http://arjuna.ncl.ac.uk/group/papers/p028.ps>

[Caughey95] S. J. Caughey and S. K. Shrivastava, “Architectural Support for Mobile Objects in Large Scale Distributed Systems,” Proceedings of the 4th IEEE International Workshop on Object Orientation in Operating Systems (IWOOOS), Lund, Sweden, August 1995.
<URL: http://arjuna.ncl.ac.uk/group/papers/p044.ps>

[Caughey97] S. J. Caughey, D. B. Ingham, and M. C. Little, “Flexible Open Caching for the Web,” Computer Networks and ISDN Systems, 29(8-13), pp. 1007-1017, Proceedings of the 6th International World Wide Web Conference, Santa Clara, California, USA, April 1997.
<URL:http://proceedings.www6conf.org/HyperNews/get/PAPER159.html> or
<URL:http://w3objects.ncl.ac.uk/pubs/fblp/>

[Chavez96] A. Chavez and P. Maes. “Kasbah: An Agent Marketplace for Buying and Selling Goods,” Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology. London, UK, April 1996.
<URL:http://ecommerce.media.mit.edu/papers/paam96.ps>

[Dublin] The Dublin Core Metadata Element Set Home Page.
<URL:http://purl.org/metadata/dublin_core/>

[Halsey97] S. M. Halsey, “The Informed Traveller: an online-service brokering system,” MSc. Computing Science Dissertation, Newcastle University, 1997.

[Ingham95] D. B. Ingham, M. C. Little, S. J. Caughey, and S. K. Shrivastava, “W3Objects: Bringing Object-Oriented Technology To The Web,” The Web Journal, 1(1), pp. 89-105, Proceedings of the 4th International World Wide Web Conference, Boston, USA, December 1995.
<URL:http://www.w3.org/pub/Conferences/WWW4/Papers2/141/> or
<URL:http://w3objects.ncl.ac.uk/pubs/bootw/>

[Ingham96] D. B. Ingham, S. J. Caughey, and M. C. Little, “Fixing the Broken-Link Problem: The W3Objects Approach,” Computer Networks and ISDN Systems, 28(7-11), pp. 1255-1268, Proceedings of the 5th International World Wide Web Conference, Paris, France, May 1996.
<URL:http://www5conf.inria.fr/fich_html/papers/P32/Overview.html> or
<URL:http://w3objects.ncl.ac.uk/pubs/fblp/>

[Ingham97] D. B. Ingham, S. J. Caughey, and M. C. Little, “Supporting Highly Manageable Web Services,” Computer Networks and ISDN Systems, 29(8-13), pp. 1405-1416, Proceedings of the 6th International World Wide Web Conference, Santa Clara, California, USA, April 1997.
<URL:http://proceedings.www6conf.org/HyperNews/get/PAPER27.html> or
<URL:http://w3objects.ncl.ac.uk/pubs/shmws/>

[Keller95] A. M. Keller, “Smart Catalogs and Virtual Catalogs,” First USENIX Workshop on Electronic Commerce, July 1995.
<URL:http://www-db.stanford.edu/pub/keller/1995/virtual-catalogs.ps>

[Khare97] R. Khare and A. Rifkin, “XML: A Door to Automated Web Applications,” IEEE Internet Computing, July-August 1997.

[Lindholm96] T. Lindholm and F. Yellin, “The Java Virtual Machine Specification,” Addison-Wesley, September 1996.

[Parrington95] G. D. Parrington et al., *The Design and Implementation of Arjuna*, USENIX Computing Systems Journal, Vol. 8, No. 3, Summer 1995, pp. 253-306.
<URL:http://arjuna.ncl.ac.uk/group/papers/p048.ps>

[Shimhoni97] S. D. Shimshoni, “QObjects,” MSc. Computing Systems and Software Design Dissertation, Newcastle University, 1997.

[Shklar97] L. Shklar, D. Makower, and W. Lee, “MetaMagic: Generating Virtual Web Sites through Data Modeling,” Poster presentation from 6th International World Wide Web Conference, Santa Clara, California, USA, April 1997.
<URL:http://poster.www6conf.org/poster/714/poster714.html>

[webMethods97] webMethods, “Web Interface Definition Language Specification,” 1997.
<URL:http://www.webmethods.com/technology/widl.html>

URLs

BarclaySquare
<URL:http://www.barclaysquare.co.uk/>

British Airways
<URL:http://www.british-airways.com/>

CDnow
<URL:http://www.cdnow.com/>

Fido the Shopping Doggie by Continuum Software, Inc.
<URL:http://www.shopfido.com/>

POET Software
<URL:http://www.poet.com/>

Railtrack
<URL:http://www.railtrack.co.uk/>

webMethods Inc.
<URL:http://www.webmethods.com/>