The Informed Traveller: A Case Study in Building Internet Brokering Services

David Ingham[1], Steve Caughey[1], Paul Watson[1], Stephen Halsey[2]
[1]Department of Computing Science, Newcastle University, Newcastle upon Tyne, NE1 7RU, U.K.
[2]Moreover.com, 4 Faulkners Alley, London, EC1M 6DD, U.K.
Dave.Ingham@ncl.ac.uk, S.J.Caughey@ncl.ac.uk, Paul.Watson@ncl.ac.uk, Steve@moreover.com

Abstract

In traditional commerce, brokers act as middlemen between customers and providers, aggregating, repackaging and adding value to products, services or information. In today's Web, such services are generally lacking with the result that individuals are forced to manually discover, collate and analyse information to meet their needs. This paper begins by presenting the design and implementation of a travel planning brokering system which provides a combined travel timetable service using information gleaned from existing Web services. The aim of this prototype was to gain experience as to the needs of Internet brokering systems in general. The lessons learned from the exercise have led to the design of a generic brokering framework, known as Metabroker. The framework 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, in the form of workflows, to support the area of speciality of the broker. Our design integrates distributed object, metadata, workflow and object database technologies.

1. Introduction

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. In contrast, today's Internet users are largely forced to struggle with a vast quantity, diversity and incompatibility of services and information. Because brokers are rare, potential customers of Internet services are forced to manually discover, collate and analyse information to meet their needs. Currently, only a limited number of Internet brokers exist and those that do have limited functionality.

We believe that one of the main reasons for the current situation is that building a broker requires significant resources and skills that are beyond many companies who would otherwise benefit from them. Because each broker must provide functionality which is specific to its particular role, then it is not possible to conceive of a single broker which could meet the requirements of the wide variety of different brokers that are, and will be, required. Therefore, a significant amount of bespoke, specialised software design and development is currently required to produce each new broker.

This paper describes an investigation into a possible solution to this problem. Commonality in the requirements of Internet brokers was identified, and a generic framework was designed to assist in their implementation and deployment. Components of the framework include: protocol handlers for communicating with a variety of types of external services; support for alternative client interaction methods; and, infrastructure support for the introduction, management and adaptation of business logic. This generic framework – which we call *Metabroker* – limits the bespoke development required to build a specific broker to that required to implement the functionality that is specific to that broker, for example its business logic. The plan of the investigation is mirrored by the structure of this paper. In order to gain experience of needs of Internet brokers with a view to the creation of a generic framework, as well as to demonstrate the usefulness and viability of Internet brokering in general, a travel planning broker (termed *the informed traveller*), was designed and implemented (Section 2). This provides a combined travel timetable service using information gleaned from two existing, independent Web services. The lessons learned, on both the strengths and weaknesses of the system, gave insights into the requirements for, and design of, the Metabroker generic brokering framework (Section 3). The power of the framework is illustrated by showing how it can be used to implement the travel timetable service. Finally we draw conclusions from our work, compare it with related work, and point to future directions.

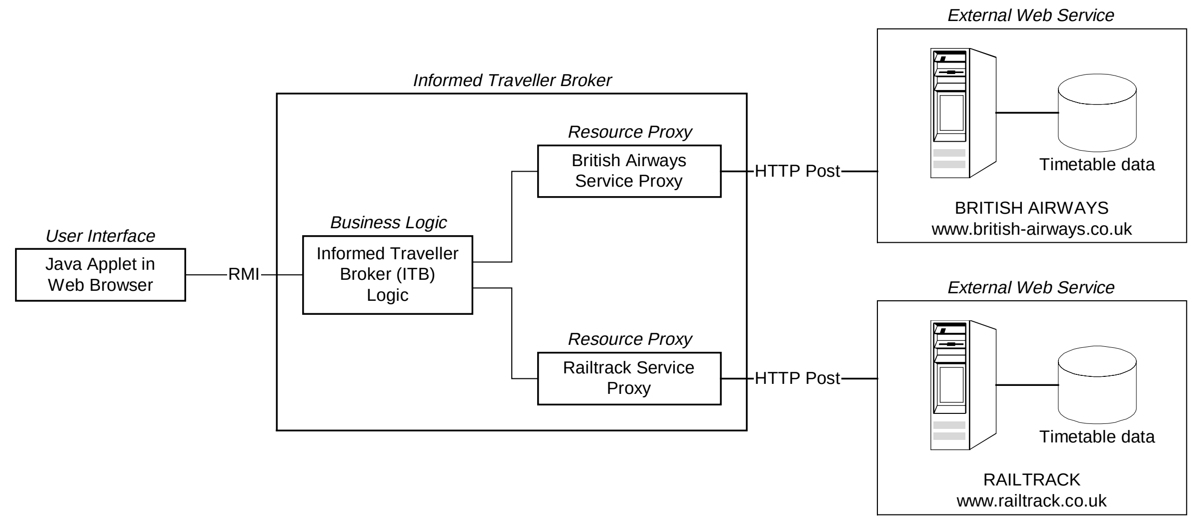

Figure 1: High level architecture of The Informed Traveller prototype

2. The Informed Traveller broker

The Informed Traveller concept was motivated by a U.K. Government Office of Science and Technology Foresight report which included a description of a futuristic travel-oriented information system that assisted travellers at the various stages of their journeys, from planning to the journey itself [1]. On the Web today there are many travel-related services on offer, including a large number of timetable services. These can assist greatly when planning single journeys, but often journey planning is complicated by the need for multi-stage journeys involving multiple modes of transport. Choosing between the available options and co-ordinating the connections is invariably time-consuming.

Our Informed Traveller broker prototype provides a small subset of the functionality described in the Foresight report. However it is non-trivial as it adds value to two existing Web based timetable services by offering information on journeys that combine two modes of transport. Constructing the prototype helped both to form and to test our ideas on the design of a generic brokering framework.

2.1. Architecture

The Informed Traveller prototype offers its services through a Web-based interface that allows the user to enter journey details, such as departure location, destination, time of travel, etc. In response to the request the service provides a unified travel plan describing the multiple stages of the journey. The queries are serviced utilising information gleaned from the online services provided by Railtrack, the infrastructure provider for U.K. train services, and British Airways.

A high level representation of the prototype is shown in Figure 1. The major components include: the user interface, implemented as a Java applet; the business logic, which provides the combined timetable interface; and the resource proxy objects that provide a programming language interface to the online services. The system is implemented in Java.

It is worth noting that, unlike in the case of many existing brokers, there is no co-operation here between the timetable service providers and the broker. Instead the Informed Traveller must interface to the existing services.

In response to a journey request from the user, the Informed Traveller creates two frames containing journey details. An "All Journeys Frame" provides a summary of the possible routes that can be taken, while the "Shortest Journey Frame" gives the detailed travel plan for the shortest of the

```
Shortest Journey is via Manchester Airport.

Overall details:
Start time:          Wed Nov 25 10:17:00 GMT 1998
Finish time:         Wed Nov 25 16:20:00 GMT 1998

Journey breakdown:
    Train journey:   1 leg
    lasting for      02 hours 52 minutes.
    starting at      DURHAM
    at time          Wed Nov 25 10:17:00 GMT 1998
    and arriving at  MANCHESTER AIRPORT
    at time          Wed Nov 25 13:09:00 GMT 1998

    Plane journey
    flight number    BA1628
    from             MANCHESTER
    at time          Wed Nov 25 14:10:00 GMT 1998
    to               AMSTERDAM
    arriving at      Wed Nov 25 16:20:00 GMT 1998
```

Figure 2: Shortest journey response frame

routes. Figure 2 shows the Informed Traveller shortest journey response for a planning request from Durham (United Kingdom) to Amsterdam (Netherlands).

2.2. Business logic

The business logic of the informed traveller is programmed in the ITB (informed traveller broker) class which also contains resource proxy objects for the two remote services. The public interface of the class includes methods that are used to populate the lists of departure and destination points. These methods basically delegate through to methods of the resource proxies without performing any significant information processing (see Section 2.3). Another method takes the user's journey details and performs the aggregation of the timetable information from the two proxies to create the unified timetable as shown in Figure 3.

- From the BA proxy obtain the list of airports that fly to the user's chosen destination.
- For each of these airports do:
- Look up the name of the train station that serves this particular airport, e.g., "Newcastle airport" is served by "Newcastle" train station. This information is held in a static table that is managed manually.
- Invoke the timetable service of Railtrack proxy for a journey from the user's departure point to the airport station just established. The date and time of travel are taken from the user's requirements.
- Add the transfer time (the default is 1 hour) to the train arrival time to calculate the earliest possible departure time for the connecting flight.
- Invoke the timetable service of the BA proxy for a flight from the airport being considered to the user's chosen destination with the timing details just calculated.
- Add the resultant journey to the list of alternative routes.
- Return the list of alternative routes

Figure 3: Combined timetable algorithm

2.3. Resource proxies

The two online services accessed by the broker provide HTML forms-based query interfaces that return an HTML document containing the results.

To simplify the programming of the business logic of the broker, proxy objects are used to abstract away from the details of how the external services are accessed. The general notion is that it is the information, rather than how it is accessed, that is of concern to the business logic of the broker. Resource proxies provide an object-oriented programming language level interface to the online services. In addition, the proxies provide transparent information caching. The Railtrack proxy (Figure 4) provides a service level interface to the business logic of the broker consisting of two methods:

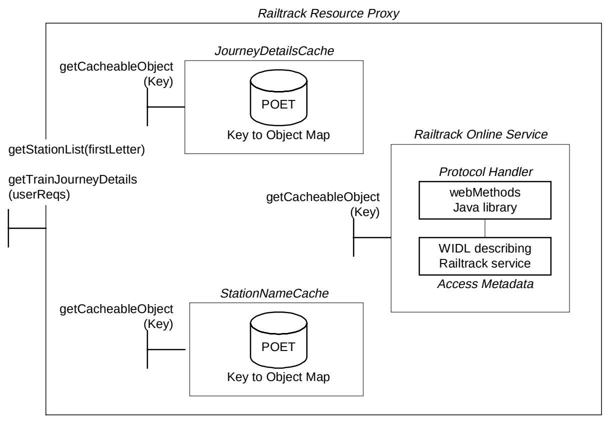

Figure 4: Resource proxy implementation for the Railtrack service

```
StationNameList getStationList (
FirstLetterOfStation firstLetter )

TrainJourneyDetails getTrainJourneyDetails (
UserJourneyReqs userReqs )
```

The `getStationList` method returns an object containing the list of station names that begin with the specified letter. This information is used to populate a list of departure point choices in the user interface. The `getTrainJourneyDetails` method returns an object containing the train timetable details corresponding to the user's requirements.

Internally, the resource proxy contains an `OnlineService` object and two `Cache` objects (Section 2.3.2). All three present the same interface which return a `CacheableObject` given an object that conforms to the `Key` interface.

2.3.1. Accessing an online service

The component of the resource proxy responsible for actually accessing the remote resource is termed a *protocol handler*. These are generic components that are able to communicate using a specific protocol, such as HTTP. A protocol handler is *bound* to a specific resource using *access metadata*. For the HTTP-based services used by the prototype, a ready-made protocol handler library, from the webMethods toolkit, was used as the protocol handler. Access metadata is defined using the Web Interface Definition Language (WIDL) [2], an application of Extensible Markup Language (XML) [3, 4].

2.3.2. Caching

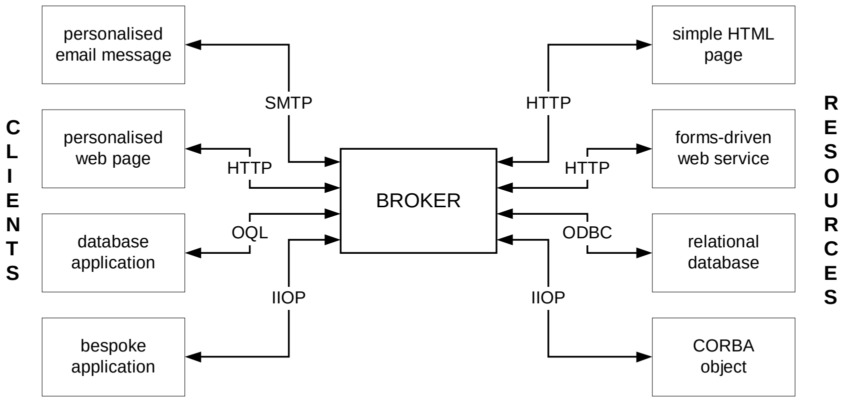

Figure 5: External broker interfaces

A single user request can result in a large number of online service invocations, and consequently a high response time. Therefore, a generic mechanism for caching data was introduced into the prototype. Although, in general, caching can be applied at a number of stages in the information path, the prototype only caches at the resource proxy level. The POET Object Database Management System (ODBMS) is used to provide the persistence support for caching in the prototype.

Objects of the `CacheableObject` base class automatically maintain information about their creation date, the number of times they have been accessed and the last time they were accessed. This allows each `CacheableObject` to independently control its cache management, i.e., determining when stored data is stale. The `Cache` object also uses this information when purging the cache to make way for new entries.

Tests confirmed the benefits of caching. Without it, the informed traveller broker can take up to 500 seconds to process a complex request, while there are significant variations in response time due to changes in the load on the resource Web sites. Caching reduces the average response time to below one second and significantly reduces the variance.

2.4. Lessons learned

The implementation of the Informed Traveller prototype was intended to give us experience in accessing remote Web services and collating the information obtained in order to produce a new service. The use of resource proxies to abstract away from the details of how information is extracted from online services proved to be a useful concept. Accessing information via the proxy using a programming-language level interface greatly simplifies the development of business logic. In particular, the use of the webMethods toolkit for HTTP resource proxies simplified our work in this area.

Building the prototype reinforced the importance of caching for brokering systems in which raw data is obtained from online services. In the prototype caching was provided at the resource proxy level only. Caching aggregated information, such as combined timetables, would further improve performance for a production system.

One of the most important lessons from this process was learnt after the prototype was complete and we began discussing how to extend the business logic. It became clear at that point that hard-wired logic, set in place at installation time, would not be appropriate for a real broker which would require frequent adaptation in order to keep pace with a rapidly changing environment.

2.5. Extending the Informed Traveller

A number of extensions to the Informed Traveller are necessary if it is to be of use in the real world. The most obvious extension is the addition of more resources representing other airlines, rail and bus companies. Additionally, many opportunities exist to broaden the scope of the system by incorporating additional Web-based value-added services, for example, travel insurance, hotel booking, etc. associated with the booking of a particular journey. Finally, a useful extension would be to maintain client information and offer personalised service.

In the next section, after describing our generic Metabroker architecture, we will show how using Metabroker would simplify the task of creating an Informed Traveller broker and providing the extensions that we have described.

3. Metabroker

The *Metabroker* brokering framework was the result of analysing the generic requirements of Internet brokers. It was heavily influenced by our experiences with the informed traveller broker. In this section we describe generic broker requirements, discuss the Metabroker design. The construction of an extended Informed Traveller broker is used as an example.

3.1. Requirements

An electronic broker utilises a set of resources supplied by external service providers in order to provide value-added services to clients. One of the key requirements of such brokers is the ability to accommodate multiple ways of interacting with clients and resources. A high level view of the external interactions of an electronic broker is shown in Figure 5. In this section we shall consider the requirements from the perspective of clients, resources, broker developers and brokering service providers.

Client requirements - Clients prefer services to adapt to their needs rather than vice versa. Therefore a broker must be capable of personalising its interactions with each client, including preferences for protocol and presentation. The broker may be able to instantly satisfy some classes of request, while others are likely to involve considerable data gathering and analysis, therefore necessitating an asynchronous mode of operation.

Resource requirements - The success of the broker will be heavily dependent on the volume and quality of the resources to which it has access. Even in a specific business domain it is unlikely that service providers will agree on standard ways to present their resources. Therefore, the broker has to be extremely flexible in its ability to interact with a wide range of protocols and data presentation formats.

Broker developer requirements - At the heart of the broker is the business logic that implements a set of services. To prove successful in the rapidly changing and complex environment that is the Internet, a broker must be highly adaptable. It must be capable of managing complex interactions between its clients and resources, each of which may consist of a number of tasks. Complex tasks such as browsing, ordering, payment and delivery might involve session-based communications and may need to be transactional. It must also be possible to introduce new functionality into the broker and to specify new, or to modify existing, business logic without causing severe disruption to the broker's operation.

Service provider requirements - A broker that 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 data to cache, and the caching policy must be configurable.

3.2. Metabroker architecture

In this section we describe the architecture of the Metabroker, which has been designed to meet the generic brokering requirements discussed earlier. Throughout this section we will draw an important distinction between the generic brokering framework that is the Metabroker, and specific brokers created by populating the Metabroker with both business logic and information on clients and resources. The overall design of the Metabroker is shown in Figure 6.

The Metabroker design is built on two key technologies: an Object Database Management Server (ODBMS), and a Workflow system. The ODBMS provides a number of key services used by the Metabroker components, including access security for information, persistent object storage and retrieval (including querying), and transactional capabilities. Workflow is the vehicle for implementing the business logic

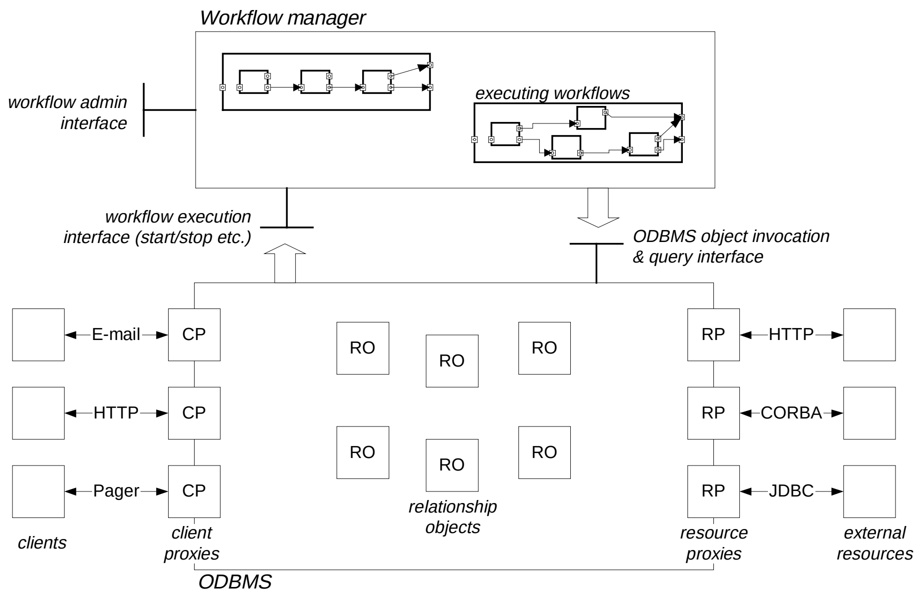

Figure 6: Metabroker Architecture

of the specific brokers created from the Metabroker. The key components of the Metabroker are now discussed in turn.

3.2.1. Client and resource proxy objects

Metabroker uses proxies for two types of external entity - clients and resources, and each proxy encapsulates the entity for the Metabroker. Proxies maintain *content metadata*, i.e. data about the content of an entity, e.g. the client's name and local station for a client proxy; the time of last modification for a resource proxy.

Every proxy object also contains *access metadata*, i.e., data about how the external entity may be accessed. This includes the protocol used to communicate with the entity, i.e., a reference to the appropriate protocol handler object together with data that is used to configure the protocol handler for the particular entities (e.g., a resource's URL). The Metabroker provides protocol handler objects for a range of common protocols, so simplifying the task of creating proxies.

One of the important features of the design is the separation of delivery, content and presentation. 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. At the resource side, this separation helps to isolate the broker (and therefore the broker's clients) from changes in the native presentation of resources. This concept is based on earlier research performed as part of the W3Objects project [5].

The Client and Resource proxy objects are stored in the ODBMS in order to benefit from persistence, transactions (e.g. to allow controlled changes to metadata) and query capability (e.g. to allow searching of metadata).

3.2.2. Business logic

Business logic is the implementation of a broker's specific services. Our design uses the Arjuna Workflow Management System [6] (its architecture was the basis of Nortel's submission to the OMG for a workflow standard [7]) in order to support the broker developer requirements detailed in Section 3.1.

A workflow is a description of business logic structured as a set of tasks interconnected by arcs that represent the flow of data. Each task can have a number of possible input sets where each set comprises a number of input objects. A task is executed whenever all of the objects in one of its input sets are present. When a task is completed it can generate output objects which travel along the inter-task arcs. Workflow programmers are free to include arbitrary application code within tasks, but within a broker tasks will commonly make invocations on proxy and relationship objects.

Using workflow to represent business logic brings two major benefits. The first is the additional fault-tolerance that may be obtained through the use of transactions within workflows and the reliable delivery of objects between tasks. The second is that, having structured the service in this way, making modifications, or extending the service, becomes a much easier process (illustrated in Section 3.3). New workflow templates can be introduced at any time, so increasing the services offered by a broker. Existing templates can be modified in a controlled manner (within a transaction) while the broker is running so that existing services can be modified dynamically.

3.2.3. Relationship objects

The workflows that implement the business logic of the broker utilise the information provided by the Metabroker objects in the database. Resource proxy objects tend to provide relatively low-level data. It is our experience that it can be valuable to support higher level objects – *relationship objects* – that provide an interface to combined information obtained from multiple resource proxies. Simple relationship objects may include hard-coded logic but more commonly are themselves implemented as workflow scripts. These higher-level objects increase the levels of abstraction and therefore simplify the construction of complex business logic. They also have the advantage that a piece of commonly required logic is implemented once (rather than in multiple workflows) which simplifies maintenance. Like proxy objects,

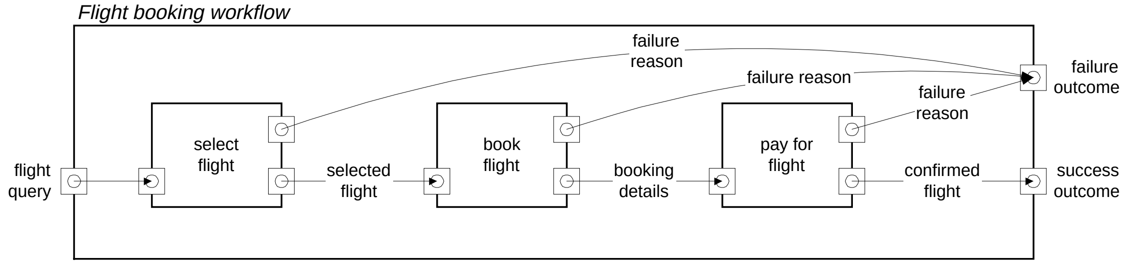

Figure 7: A flight booking workflow template

relationship objects can utilise Metabroker caching services.

3.3. Implementing an Informed Traveller Broker using Metabroker

In this section we consider how the use of the Metabroker framework can be used to simplify and extend the construction of an Informed Traveller broker such as that described in Section 2.

In our prototype Informed Traveller the business logic of the service was implemented by an ad-hoc piece of code. In contrast, a Metabroker based system would implement services as workflows. An example workflow for a flight booking service (much simplified for the purposes of illustration) is shown in Figure 7.

This separates out three distinct tasks in the flight booking process: `select flight`, `book flight` and `pay for flight`. Tasks perform their function by invoking operations on Metabroker proxy and relationship objects. For example, `pay for flight` invokes `the bank gateway` in order to debit the client's account as shown in Figure 9.

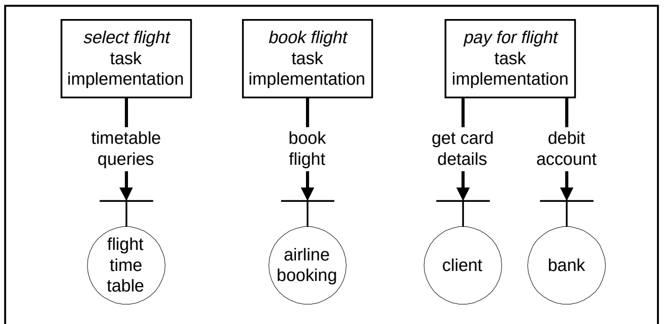

Figure 8: Task / Metabroker object interaction

One way of extending the Informed Traveller, discussed earlier in Section 2.5, involved the addition of travel insurance services. Figure 9 shows how an insurance workflow could be incorporated into the flight booking workflow, shown in Figure 7. The insurance workflow is added by connecting the `booking details` object reference that is output from the `book flight` task to the input of the insurance workflow.

Although the prototype utilised resource proxy level caching to improve performance, a Metabroker implementation could use relationship objects to provide combined travel information from several sources, e.g., airline, rail and bus companies. Caching at this level would further improve the response time as seen by clients.

The user interface of the informed traveller prototype was inflexible, supporting only a synchronous Java applet-based interface. Considering that some broker functions could take several minutes to perform, perhaps email notification of the results would be more appropriate. The client proxy concept of separating presentation logic from functional interface would support this flexibility.

4. Related work

The Metabroker design builds on the distributed object technologies produced by the Arjuna distributed systems group of Newcastle University. Most significantly, Metabroker uses the Workflow Management System for structuring business logic (this internally uses Arjuna transaction services). Additionally, the W3Objects research focused on the management of Web-based applications through the separation of presentation logic from functionality has fed into the design of our client proxy concepts.

The Arjuna technologies are already being applied to meet

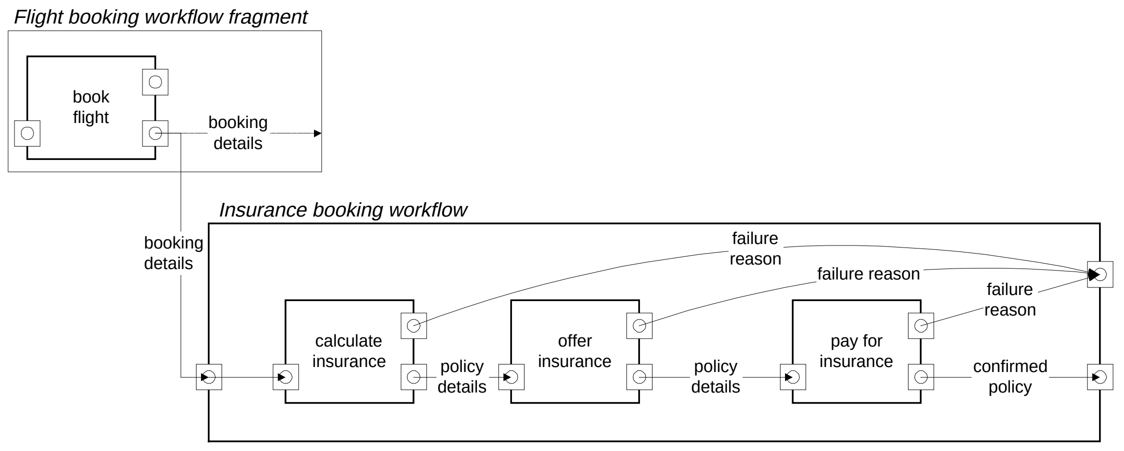

Figure 9: A travel insurance workflow template

the needs of businesses working in the area of electronic commerce. For example, within the context of the ESPRIT MultiPLECX project we are working with TradeZone International to use workflow to support the complex business processes that are inherent in business-to-business electronic commerce.

BarclaySquare (an online shopping mall), TradeZone (a catalogue service) and the Global Electronic Music Marketplace are examples of the large number of existing commercial services that offer brokering functionality. However, all of these services differ from our work in that they are more centralised in nature; information and services provided are maintained centrally using data formats and protocols that are under the control of a single organisation. In the specific area of travel services, A2bTravel, provide a convenient service that encompasses flight, rail, and bus service planning and booking. However, the timetable services offered are self-contained; there is no attempt to provide a combined travel planning service similar to the Informed Traveller. The Metabroker framework is designed to support brokers that aggregate and add value to information or services that are outside of the broker’s control.

Our approach is similar to that adopted with the Smart Catalogs architecture [8] 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, only one example of which could be a smart catalogue.

MetaMagic [9] 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, Metabroker extends the model to include clients and supports complex services implemented as workflow.

Agent-mediated brokers, such as Kasbah [10], use communicating agents to represent the clients and services interacting within some market. In our system the agents are workflows. These can be long-running when some complex activity is to be performed and can be initiated periodically or when external events impinge upon the system. As such they may be seen as acting autonomously on behalf of their owner. Workflow is a useful, and well-understood, infrastructure within which to manage and dynamically interconnect agents.

5. Conclusions

In common with others [8, 10], 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. Therefore, our work has focussed on the design and construction of these electronic brokers. Our approach, which has been mirrored by the structure of this paper, is as follows:

1. Design a prototype broker to gain experience with the requirements and technology.
2. Specify the set of generic requirements for brokers, i.e., requirements that are independent of their particular application domain.
3. Design a generic brokering framework – Metabroker – that simplifies the task of designing and building specific brokers by providing common facilities from which specific brokers can be built.

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. However, we need to represent these diverse external entities as uniform internal entities that can be manipulated in a consistent fashion within the broker. Metabroker achieves this through the use of client and resource proxies. A broker must also be responsive to changes in its external environment which offer new business opportunities. The use of workflow as a central Metabroker component allows the business logic to be evolved and extended in order to exploit such opportunities.

The experience gained in implementing prototypes has convinced us of the necessity and viability of the Metabroker approach [11]. We are utilising components from these prototypes together with the Arjuna distributed object and workflow services [5, 12, 13] to implement the full Metabroker architecture. This will then be used to build exemplar brokers to further test our ideas.

Acknowledgements

The work reported here has been partially funded by the European Union as part of ESPRIT project number 26810, Multi-Party Processes for Large-scale Electronic Commerce Transactions (MultiPLECX).

References

[1] “Progress Through Partnership: 5 Transport,” U.K. Government Office of Science and Technology ISBN: 0-11-430116-6, March 1995. <URL:http://www.foresight.gov.uk/itec/docs/ptp5/attach.html>

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

[3] R. Khare and A. Rifkin, “XML: A Door To Automated Web Applications,” *IEEE Internet Computing*, vol. 1, no. 4, pp. 78-87, 1997.

[4] T. Bray and S. DeRose, “Extensible Markup Language (XML): Part 1. Syntax,” World Wide Web Consortium, Working Draft (work in progress)

November 1997.
<URL:http://www.w3.org/TR/WD-xml>

[5] D. B. Ingham, S. J. Caughey, and M. C. Little, “Supporting Highly Manageable Web Services,” *Computer Networks and ISDN Systems*, vol. 29, no. 8-13, pp. 1405-1416, 1997.
<URL:http://w3objects.ncl.ac.uk/pubs/shmws/>

[6] S. Wheater, S. K. Shrivastava, and F. Ranno, “A CORBA Compliant Transactional Workflow System for Internet Applications,” presented at IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), The Lake District, England, 1998.
<URL:http://arjuna.ncl.ac.uk/group/papers/p076.pdf>

[7] Nortel and Newcastle University, “Revised Submission for Workflow Management Facility Specification,” OMG Document bom/98-03-01, 1st March 1998.
<URL:http://www.omg.org/>

[8] A. M. Keller, “Smart Catalogs and Virtual Catalogs,” presented at First USENIX Workshop on Electronic Commerce, 1995.

[9] L. Shklar, D. Makower, and W. Lee, “MetaMagic: Generating Virtual Web Sites through Data Modeling,” presented at Sixth International World Wide Web Conference, Santa Clara, California, U.S.A., 1997.
<URL:http://poster.www6conf.org/poster/714/poster714.html>

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

[11] S. Halsey, “The Informed Traveller: an online service brokering system,” in *Computing Science*: Newcastle University, 1997.

[12] S. J. Caughey, “SHADOWS: A Flexible Support System for Objects in a Distributed System,” presented at 3rd IEEE International Workshop in Object-Orientation in Operating Systems (IWOOOS), Ashville, North Carolina, U.S.A., 1993.
<URL:http://arjuna.ncl.ac.uk/group/papers/p028.ps>

[13] G. Parrington, “The Design and Implementation of Arjuna,” *USENIX Computing Systems Journal*, vol. 8, no. 3, pp. 253-306, 1995.
<URL:http://arjuna.ncl.ac.uk/group/papers/p048.ps>

URLs

A2bTravel
<URL:http://www.a2btravel.com/>

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

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

ESPRIT Project MutliPLECX
<URL:http://www.multiplecx.org/>

Global Electronic Music Marketplace
<URL:http://www.gemm.com/>

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

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

TradeZone International
<URL:http://tradezone.onyx.net/>

WebMethods, Inc.
<URL:http://www.webmethods.com/>