A Broker Architecture for Integrating Data Using a Web Services Environment

K.H. Bennett[1], N.E. Gold[2], P.J. Layzell[2], F. Zhu[1], O.P. Brereton[3], D. Budgen[3], J. Keane[2], I. Kotsiopoulos[2], M. Turner[3], J. Xu[1], O. Almilaji[2], J.C. Chen[2], and A. Owrak[2]

[1] Department of Computer Science, University of Durham, Durham, UK, DH1 3LE
{keith.bennett, jie.xu, fujun.zhu}@durham.ac.uk
[2] Department of Computation, UMIST, Manchester, UK, M60 1QD
{nicolas.gold, paul.layzell}@co.umist.ac.uk
[3]Department of Computer Science, Keele University, Keele, UK, ST5 5BG
{d.budgen, o.p.brereton}@cs.keele.ac.uk

Abstract. The web service protocol stack provides capabilities for loosely integrating software services but does not provide the higher level support needed for rapid evolution. An experimental system is described for integrating the data from autonomous organizations within the UK health service domain. The results of this experiment have confirmed the need for an integration layer on top of the web service stack to provide the required higher level functionality. In this paper, we summarise our progress to date, and highlight several key research issues of general concern to the web services field, which have emerged from our prototype system. These are set in a general context of providing better ways to provide a service-based model to IT users.

1 Introduction

The web service protocol stack (SOAP, WSDL etc) provides capabilities for loosely integrating software services. Currently, such protocols are limited to providing communication and basic synchronization between services but do not support the higher-level functionality needed to enable dynamic and ultra-late binding of services to form software applications on demand to a specific set of requirements. Over the last decade, we have been investigating ways to produce highly-evolvable software, resulting in a service-oriented approach to software construction. This approach requires far more than the current web service protocol stack provides, specifically, an integration layer allowing the interpretation of requirements, negotiation of terms, conditions and price, service failure management (including warranty and redress) and management of the service supply chain. These issues were identified in early prototype implementations [8] and have been re-confirmed by our experience of building the prototype described here. The issues we identify provide a roadmap for the development of the web service stack.

M.E. Orlowska et al. (Eds.): ICSOC 2003, LNCS 2910, pp. 409–422, 2003.

2 Background to Our Research: "Software as a Service"

In the 1990s, the Pennine Research Group (software engineers from the universities of Durham, Keele and UMIST) together with members of the British Telecom Research Laboratories undertook a project to consider the future of software over a ten year timespan. The main findings [8, 9, 10] are expressed in terms of six key themes. Our approach was to address software from a user perspective, and a summary major conclusion was that the cost of ownership needs to be reduced. For example, users were very unhappy with having to evolve, install, and support software (whether bespoke or shrink wrapped). These disadvantages arise because software is owned, whereas most users simply require the results from using the software.

Our conclusion was that software needs to move from being supply-side led to demand-side led; in other words, software becomes something that is used, not owned. The choice of competitive services within a marketplace is up to the user (or their applications) based on qualities such as cost, warranty and performance. We coined the term "software as a service" to describe this (see www.service-oriented.com). A service-based model is one in which one or more services are configured to meet a specific set of requirements at a point in time, executed and disengaged [10]. This view is an example of a demand-led software market in which software possession and ownership is separated from its use. This allows alternative services to be substituted between each use of a system, allowing much richer finer-grained flexibility. The data becomes available on demand and as available without perturbing the operational systems.

We identified the importance of indirection and ultra-late binding (at the point of need) in a large-scale distributed system to support this model. This is necessary but not sufficient. As a simple example, a service application on machine A, wishing to use a service on machine B, will need to pay for it, and in doing so expect some obligations in return (such as quality of service, warranty, and performance). Binding to such information will need to be automatic, without human assistance. Typically, agreement of such costs will require negotiation between parties. Thus software as a service involves issues far wider than the simple late binding of services using standard protocols. Many of the new problems occur in this area, which we call "terms and conditions".

Such terms and conditions arise because service based software is being used in a business environment. It is unrealistic to expect or require that software engineers invent a whole new environment for doing business according to a whole range of business models. Instead we expect that existing models will be used, of which there is of course a vast experience acquired over thousands of years. The contribution will therefore be to understand how these existing solutions are adapted, represented and used in the programmatic interfaces offered by service based software. Solutions are expected to be interdisciplinary, and a group, ISEN, has been set up to foster research (www.service-oriented.com/isen/).

During this research, a significant development in web services technology took place, which has become part of the enabling technology baseline by which part of our vision for future software could be realised.

Shirky [6] suggests that web services address application program inter-operability. The most general form involves binding complex programs together from pieces

anywhere in the world. General inter-operability has been tried before, but with partial success, for example DCOM, Corba, and RMI. With these systems, both the client and server have to load the system; with web services, the idea is to know nothing about the "other end" other than what can be communicated via standard protocols. So WSDL allows the description of a service so that a call for it can be assembled and invoked from elsewhere. In order to communicate data in a system independent way, XML is used. A UDDI registry allows service vendors to offer services, and users to locate and call them using WSDL descriptions published along with service identification information.

However, these web service technologies are really only the first part of the solution to more flexible software. At the end of the initial phase of work, our conclusion was that for web services to be widely used, a good technical solution alone is insufficient. A major factor in the widespread acceptance of service-based approaches is to provide an architectural layer beyond the existing web service technology which provides added-value in terms of a service supply-chain, a services market and appropriate levels of trust, confidence and importantly, security [11, 12]. Such a layer can be regarded as an *integration* layer in a services technology stack.

Software has previously been developed, delivered and maintained as a product. The internet is stimulating interest in software which is instead delivered and used on demand, because this potentially allows faster and more flexible evolution to meet changing business needs; there is a much looser coupling with a service based approach between business requirements and software solution. However, actually implementing this is far more complex than technical considerations alone would suggest. For example, service based software needs to be identified and then selected; this may well require negotiation within a market [7]. The consumer application will need to have confidence that the service performs as described, and if not, means of redress are available. Fundamentally, the service model will fail if there is a lack of trust between vendors and users. Our overall research is therefore directly concerned with these *wider* problems.

To test our research findings and to scope the requirements for an integration layer in the web services stack, we have undertaken a prototype implementation of web services, addressing a real life problem. We choose, as the prototype application domain, the issue of *integrated health care data*. This was a highly appropriate case study because it combines the need for flexible software (functionality) with issues of independent, heterogeneous data sources (data) which similarly need the type of ultra-late binding required by the system's functionality.

3 The Experimental System

3.1 Application Domain

We aim to support decision-making processes where multimodal information is drawn from a set of heterogeneous, autonomous agencies. The domain of health and social care has been chosen as it offers practical examples of all the problems for which we seek technical solutions.

The UK approach is basically imposed, top down, large IT systems which encompass all the contributing organisations, ranging from general medical practitioners, through acute hospitals, and include specialist services such as pathology. Additionally in the UK, the social services are included, because of the desired aim to produce "seamless" service for the hospital patient who is then discharged to the care of local social services.

One approach is *data warehousing* [1] where operational data sources are collected into a large, centralised data store. This 'filing cabinet' approach has the disadvantages of data duplication, and update issues. Further complications arise when the underlying data is 'owned' by different organisations and is confidential. Large-scale fully integrated IT systems have failed repeatedly to provide solutions possibly because these systems were difficult to construct, manage, and evolve. Therefore, in a rapidly evolving environment alternative solutions have been proposed to enable adaptation to continuous change and maintenance of the trustworthy requirements related to ownership of confidential data, whilst enabling the global view of the distributed data sources.

Primary care practices, hospitals, mental health trusts and community health services are independent organisations, each with their own information systems and with strict rules about who may access information. Treatment of a patient will however require access to *all* records relevant to the case. Our aim is to explore the integration of data from many sources, given its often heterogeneous nature, for example, in terms of such aspects as format, semantics, meaning, importance, quality, ownership, cost and ethical control.

When information is created, modified, and stored independently, its integration requires run-time binding on demand. In human-centred management of information, this role is often performed by a human broker (such as a travel agent), who is able to integrate information on demand. In this project, we are employing a similar model, and the purpose of IBHIS (Integration Broker for Heterogeneous Information Sources) is to create an *information broker service* that will support the reliable integration of information held and managed by heterogeneous autonomous agencies. The potential for this approach can be found in many domains as well as healthcare (travel, military command & control, entertainment etc.). The project is being evaluated through the development of a series of prototype brokers and proof-of-concept healthcare demonstrators.

Brokerage, where distributed, heterogeneous data sources act as a global resource for the purposes of querying associations between and within sources, allows existing data sources to continue operational activity on their data and to retain ownership of that data. Brokerage builds on both information integration and interoperability in order to provide mediation to resolve impedance at multiple levels [2].

Potential advantages of a broker approach against fully integrated systems include:
–Supports *multiple, independent* data sources
–Handles *syntactic, semantic* and *system heterogeneity*
–Deals with *globally distributed information*
–Provides a *pathway* towards discovery and access of new information resources with the minimum of human intervention.

The broker seeks to give the user a customised, virtual picture of key information, when and where it is needed, using with permission data from autonomous systems.

There are different approaches to the problem of integration of heterogeneous information sources depending on various constraints such autonomy, security, level of integration, performance etc. Agent based systems, knowledge-based information brokers, web information retrieval and brokering systems and Federated Database Systems are all different implementations of the mediated approach. A detailed survey of such systems is published by Paton et al. [3].

Several mediated systems or information brokers have been applied in the health domain. WebFINDIT [4] is targeted at medical research in hospitals. Syanpses and SynEx [5] provide integrated views of patient data from heterogeneous, distributed information systems using the federated approach.

The health service is a highly complex domain, and for the purposes of our project, it was decided to focus on three key research areas:

1. The extent to which broker architectures can support the integration of heterogeneous data to given a single view of data for a patient.
2. How security and privacy should be addressed within a strong ethical context. Such properties have to be provided with a very strong audit function.
3. To what extent the evolution of health service IT can be supported.

In order to gain a good understanding and model of the domain, extensive activities have been undertaken involving health service professionals; these are not reported here.

3.2 Architecture

Our approach uses a three stage experiment, of which we report in detail on stage (a) which has been implemented:

a) a federated schema approach using a passive broker with data access services.
b) a service based approach using a passive broker.
c) a service based approach using an active broker.

A "passive" broker seeks information on demand and offers a customised view of data. An "active" broker notifies the user of key changes. Our basic concept is that the user queries IBHIS, and IBHIS interrogates 'local' data access services, and coalesces results. Initially, this is based on statically-bound set-up knowledge of data sources where the frequency of change may be critical; IBHIS uses a traditional federated schema solution to integration. In the next version, this will be replaced by a broker which itself is a service and can dynamically locate and bind data sources which are not compile time fixed. The architecture of the first IBHIS broker is an amalgamation of FDBMs, ontology use, and service-based software

The service-based model is realised in the first phase architecture by using web services and open standards and protocols such as Java, SOAP, WSDL and UDDI. At the same time, the global view of the distributed data is achieved by the creation of one or more federated schemas according to the user requirements. Data source registration and schema integration are essentially assumed to occur 'once-and-for-all' at set-up time. Within the Health and Social care domain this is adequate as a

prototype; the possibility of data sources becoming unavailable during periods is catered for. IBHIS currently operates within and between a relatively small number of data sources, all of which hold 'real' data. In a more realistic model, it is necessary to design a supply chain, where a hierarchical structure is used, and the potential for one of the data sources itself to an IBHIS operating in a neighbouring health or social care authority. In our experiments, only artificial data about imaginary patients is used.

This notion of a 'meta-IBHIS' lends itself potentially to a scalable architecture, but the efficacy of this model across many hundreds of data sources is an area of investigation.

The users of the IBHIS broker are provided with transparent access to the heterogeneous, distributed data access services once the set-up phase is complete. During the set-up, the registration of the users and the underlying data services takes place. The system (or federation) administrator constructs the federated schema and resolves all the semantic differences. The data recorded or created during the system set-up are passed using XML to the Operational System. The architecture of the IBHIS broker is described below and in Fig. 1.

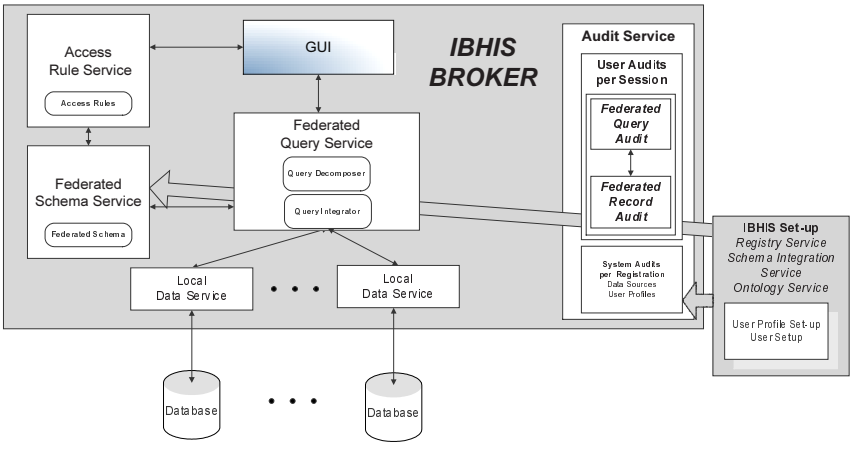

Fig. 1. The IBHIS Broker

3.3 Operational System

The aim of the operational system is to acquire a query from the user, identify access rights, locate the appropriate information sources and return the results to the user. In order to provide this functionality, the operational system consists of five communicating web services and a user interface. The web services interact according to the following model [13]:

- A data access service advertises its WSDL definition into a UDDI registry which in our case is integrated in the registry service.
- The client looks up the service's definition in the registry.

- The client uses information from the WSDL definition to send messages or requests directly to the service via SOAP.

Access Rule Service (ARS). The ARS is responsible for the initial user authentication and subsequent authorisation. Within the architecture, the ARS is primarily concerned with authorising access to the available data access services. Authorisation to other system resources is performed by the services themselves.

The initial authentication is based around usernames and encrypted passwords. The user logs onto the front-end, which in turn passes the user's credentials to the ARS. The current solution uses role based access control, where each role has its set of access rights. Much research has been conducted into Role Based Access Control (RBAC), and specifically into how it can be applied within the Health domain. However, this work has indicated that RBAC alone was too inflexible and generalised for use within the system architecture and so a more complex set of user-based access rules were developed. These, along with the access rights for the corresponding role, are used to form a user profile. In conjunction with the Federated Schema Service, the user profile identifies which aspects of the federated schema the user is authorised to view.

Federated Schema Service (FSS) and Query Service (FQS). The Federated Schema Service (FSS) keeps the federated schema and all the mappings between the export schema and the federated schema. The FSS is consulted by the Federated Query Service during the query decomposition and integration process.

The Federated Schema and the corresponding mappings to the Export Schemas are created during the set-up of the IBHIS broker.

The FQS is comprised of two sub-modules:

- *Query Decomposer:* decomposes the federated query into a set of local queries; this is done in consultation with the FSS.
- *Query Integrator:* receives the set of local results from the data access services and it integrates them into a federated record.

The FQS sends the federated query and the federated record to the audit service; this is comprised of two sub-modules which keep track of every action of IBHIS that needs to be recreated or audited in the future.

- *User Audit* (per session): holds information such as: user log-in date, time, IP, logout, sequence of federated queries, of federated record, of sessions, etc.
- *System Audit* (per Registration): holds information about data source (e.g. registration date and time, intervals of availability, etc) and user setup.

Data Access Service (DAS). The DAS is constructed using web services, but unlike typical web services, the DAS are data intensive and are responsible for providing data from their respective sources. The broker administrator implements and describes the service using WSDL and the Web Services Policy Framework [14] and also provide the consumer of the service (the FQS) with the following information:

- The data that the DAS provides, and its format
- The domain and functionality related to the data,

- The security requirements for using the service
- Other non-functional characteristics, including quality of service, and cost

The administrator then publishes the description file into the registry service, for discovery at run-time by the IBHIS operational system. For example, the DASs themselves may be programmed using different languages, and may access data sources produced by different vendors or may run on different operating systems, but the DASs provide a unified way to access the data. This is essential in the UK health services where data sources derive from many autonomous organisations, and use a range of different technologies.

When the FQS decomposes the federated query into a set of local queries, the FQS uses the registry service to look for a corresponding DAS that provides the required data outputs for each sub-query. It then uses the DAS description to bind with the data service, which accesses the data source owned by, for example, a local hospital.

A detailed study of suitable implementation tools and environments was undertaken. We are currently using Sun's Java 2 Platform, Enterprise Edition (J2EE) and IBM's Websphere.

The experimental system comprises three databases holding data about imaginary patients:

–Basic Patient Information (Keele)
–Treatment History (Manchester)
–Further Appointments (Durham)

There are three users with different authorisation levels according to their role.

This first prototype has achieved the following: a substantial application has been built using web services. Familiarity has been gained with web services toolsets and environments, providing reassurance on the technology. Extensive understanding of a highly complex domain has been achieved. The basic idea of a broker has been implemented.

This prototype has been useful in re-confirming our identification of the major problems which are discussed further in the next section.

4 Discussion and Results

4.1 Architectural Issues

The prototype implementation represents one particular approach on a spectrum of static binding to very late dynamic binding. A federated schema structure was employed, built on component data access services at three sites. This showed that creating such a high level schema manually from several data access services was feasible, provided that the number of data access services is small, and the data access services do not change often. If the approach is to be scaled up, manual integration of the federated schema (at design time), based on rapidly changing component data access services is not viable. A clear research problem is to understand better the trade-offs in achieving late binding (on demand) of data.

4.2 Web Service Protocols

As far as possible, the prototype employed standard web services protocols. The use of SOAP was successful. The prototype used RPC protocols for client-server interaction. This proved simple to implement, and offered good performance for simple calls. On the other hand, it enforced rigid data typing and binding. In retrospect, the use of document access protocols would have allowed a query to the data access service to be constructed dynamically. The RPC mechanism forced us to use artificial methods of parameter transmission using packed strings instead of arrays (hence requiring an overhead to pack, and then parse).

We also found that the tight coupling inherent in RPC calls caused severe problems during the development of the prototype, as service interfaces changed rapidly. It is not likely that RPC will provide an adequate mechanism for inter-program service calls in the face of rapid evolution. Our prototype was able to cope because we specified the service interfaces in advance. This is a serious restriction, and in subsequent prototypes we will abandon this and experiment with document messaging.

4.3 Registry

In the prototype, a UDDI registry was not used; as a result, the broker needed to know about component data access services and interfaces at design time, although it still enabled particular data access services to be located and bound in at run time (given that the broker has a built-in extensive knowledge of the services). Clearly this does not allow new data access services to be added easily, or permit changes to existing services. A priority in the next prototype is the addition of a full UDDI type registry for the support of dynamic data access services. This is a significant research problem, as most web service experiments are concerned with providing executable code as a service, not data. For example, it is not clear if the interface to the data access service should be procedural (perhaps including in the service some business logic); an alternative is to provide an interface based on SQL.

4.4 Web Services Description

We used WSDL exclusively to describe our web services. This was sufficient to define adequate descriptions for the prototype, but we do not feel it will be sufficient to describe the full functional and non-functional attributes necessary for a realistic service oriented approach. In particular, it does not provide an adequate level of description of security, versioning, quality of service, and costs. More generally in the health service environment, we see that ontology-base approaches will be essential to map between the various schemes of terminology in use; certainly, the keyword type access of UDDI will be insufficient.

4.5 Development Issues

The three distributed data access services were programmed using different concepts, at three different sites, on heterogeneous platforms, and based on different data base management systems.
The project used IBM's Websphere for the prototype. The services constructed met the aim of platform and language independence, and we are confident that (following experiments with GLUE, a second development environment) that implementation independence should be readily achievable.
We found that Websphere (V5) provided the facilities to implement, test and deploy web services. The ability automatically to generate WSDL, SOAP and XML saved much time. The Concurrent Versions System supported version control for development distributed across three sites.

4.6 Summary

The use of an integrated development environment saved considerable time, though there was an inevitable overhead in familiarisation. Otherwise, two central issues have emerged from our prototype:

1. Potentially, web services can benefit from very late binding in order to construct, on the fly, a system which is required by the user. On the other hand, the achievement of dynamic binding is beyond the capabilities of current protocols. The trade-offs and capabilities on this spectrum of static through to ultra-late binding need further experimentation and deeper understanding.

2. Much work on web services has concentrated on functional provision. It is clear that additional problems of description, performance, scale, privacy and level of abstraction arise with services which focus on data.

5 Wider Issues

Our experience with the IBHIS project has additionally confirmed that many of the wider issues in service-based software that we have identified in previous work [8, 9, 10] exist for data-oriented service integration as much as for functional service integration. This section describes these in more detail.

5.1 Supply Chain Formation and Management

The potential issues about supply chain management have been recognised in component-based software engineering (CBSE) environments. However, most of the solutions suggested are largely confined to using repositories or documents to indicate component attributes [15, 16, 17, 18]. Such solutions favour stable user requirements which are unlike the rapidly changing nature of the user requirements in IBHIS

environments. On the other hand, web services only address the technological issues of the problem caused by heterogeneity among enterprise applications, middleware, and components. Web services do not solve the problem of rapidly changing requirements and the vast dynamics in supply chains (essentially recursive use of web services) caused by requirement changes.

In a service environment, software services will be procured and assembled from sub-services or components along supply chains. Therefore, optimising such supply chains is essential. One question emerges: are the supply chains visible for all participants in order to extend optimisation and coordination? A negotiation description language (NDL) provides part of the answer to this question. Information stored in automatic negotiations along supply chains can be very useful for other participants to adjust their own activities for the purpose of optimising the whole performance of supply chains. So far NDL is specific to one-to-one negotiation situations [7, 19]. By expanding the scope of NDL to many-to-many negotiations, participants can simultaneously obtain information about other suppliers and react accordingly. More importantly, the information exchanged in negotiations can cover both technical conformance and other legal, commercial aspects. Such information will determine the success of online marketplaces in service environments. Successful negotiation results guarantee not only the conformance of technical requirements but also the agreement and fulfillment across various non-functional issues such as prices, terms and conditions.

The central theme of our current research in this area is to combine the aspects of many-to-many negotiations in NDL and optimize them using techniques such as Quality Function Deployment [20]. With this approach, a user can go to the marketplace to post requests for software services and have needs satisfied. Service providers in the supply chain responsible for the request can procure, assemble, and deliver their promised service according to agreed price, time duration, or other kinds of technical or contractual conformances. This approach could allow service providers to manage their service offerings and the according supply chains in an efficient and viable way.

5.2 Service Quality Assessment

A new process is needed to address the issue of software quality. This would form a *just-in-time* audit agent rapidly to assess the quality of individual software services prior to system composition. Such a process would need to satisfy all of the usual characteristics that exist within product quality evaluation, whilst addressing the specific needs that are essential to the delivery of services. Furthermore, the development of such a process will allow system brokers and consumers to identify the important quality features that are relevant to their needs prior to system composition. We are developing an automated tool, capable of performing quality assessments on the fly.

We are concerned with two aspects of quality. The first focuses on what attributes and characteristics of quality can be identified within our “software as a service” model. The second addresses the measurement of these quality characteristics with the aim of allowing automatic compatibility assessments of services to take place.

Selecting the 'best' service is a complex decision process for clients. The process requires a large number of quality characteristics to be simultaneously measured and evaluated. Many of these are related to one another thus may conflict insofar as improvement in one often results in decline of another (e.g. as usability increases, security may decrease).

We have devised a quality model linked to the International Standard for Software Product Evaluation ISO-9126 (see [21]). The model aims to support the automated evaluation of services prior to service composition.

5.3 Minimising Composition Time

One factor crucial to the success of our model is the speedy composition and delivery of services to the customer. Two major factors are:

1. *Searching time*: the time spent on the process of screening the marketplace to find the required service's components.
2. *Composition time*: the estimated time spent on assembling the combination of selected components. Composition can be defined as the binding of many services into new service that is expected to satisfy the user requirements and be ready to run. Based on that, the composition phase of producing service-based software could be seen as equivalent to the design phase in traditional software engineering.

In our current composition model, the selection of a service's components will depend on the behaviour or functions required while the form or style of the composition will rely on the communication. Thus, the style of the composition which will be carried out will consider only the communication (interface, connectors, data and the relationships) specified between the service's components. No knowledge of the internal structure of the service's components is used to derive the composition.

5.4 Comprehending Service-Based Systems

Another aspect of our work is looking at mechanisms, information, and processes for handling software failure in a service-oriented architecture. This work is at an early stage but thus far we have only theoretical discussions of how such a failure might be comprehended.

6 Conclusion

We have described the first stage of an implementation of a service based solution to integrating heterogeneous independent databases, using a broker architecture. The IBM Websphere and J2EE systems have been used as the testbed. We have been able to draw wider conclusions about the appropriateness of our approach for larger scale systems which include both data and functional services. In terms of the three research questions posed in 1.1, we have implemented a solution to the first two used web services. We have not yet addressed the third issue (evolution) and this is now the focus of our research.

Acknowledgements. The authors would like to thank all members of the IBHIS project and the Pennine Group for their valuable contributions to this paper. The financial support of EPSRC through the DIM programme is acknowledged**.**

References

1. Widom, J., Research problems in data warehousing, Proceedings of 4th International Conference on Information and Knowledge Management, (1995)
2. Kashyap, V. and Sheth, A., Information brokering across heterogeneous digital data: a metadata-based approach. Boston; London: Kluwer Academic (2000)
3. Paton, N.W., Goble, C.A., and Bechhofer, S., Knowledge based information integration systems, Information and Software Technology, Vol. 42, No. 5 (2000) 299–312
4. Bouguettaya, A., Benatallah, B., Ouzzani, M., and Hendra, L., WEBFINDIT: An architecture and system for querying web databases, IEEE Internet Computing, vol. 3, No. 4 (1999) 30–41
5. Grimson, J., Stephens, G., Jung, B., Grimson, W., Berry, D., and Pardon, S., Sharing health-care records over the Internet, IEEE Internet Computing, Vol. 5, No. 3 (2001) 49–58
6. Shirky, C., Web services and context horizons. IEEE Computer, September, Vol.35, No. 9 (2002) 98–100
7. Layzell, P. J. and A. Elfatatry, Negotiating in Service Oriented Environments. Comm. ACM. (to appear).
8. Bennett, K. H., Gold, N. E., Munro, M., Xu, J., Layzell, P. J., Budgen, D., Brereton, O. P. and Mehandjiev, N. Prototype Implementations of an Architectural Model for Service-Based Flexible Software. Proc. Thirty-Fifth Hawaii International Conference on System Sciences (HICSS-35), edited by Ralph H. Sprague, Jr. Published by IEEE Computer Society, CA, ISBN 0-7695-1435-9 (2002)
9. Bennett, K. H., Layzell, P. J., Budgen, D., Brereton, O. P., Macaulay, L., Munro, M., Service-Based Software: The Future for Flexible Software, IEEE APSEC2000, The Asia-Pacific Software Engineering Conference, 5–8 December 2000, Singapore, IEEE Computer Society Press (2000)
10. Brereton, P., Budgen, D., Bennett, K.H., Munro, M., Layzell, P.J., and Macaulay, L.A., The Future of Software: Defining the Research Agenda, Communications of the ACM, Vol. 42, No. 12 (1999)
11. Yang, E.Y., Xu, J., and Bennett, K.H., A fault-tolerant approach to secure information retrieval, in Proc. 21st IEEE International Symposium on Reliable Distributed Systems, Osaka, Oct. (2002)
12. Yang, E.Y., Xu, J., and Bennett, K.H., Private information retrieval in the presence of malicious faults, in Proc. 26th IEEE International Conference on Computer Software and Applications (COMPSAC2002), Oxford, Aug. (2002)
13. Vinoski, S., Web services interaction models, part 1: Current practice, IEEE Internet Computing, vol. 6, No. 3, (2002) 89–91
14. IBM, Microsoft, BEA, SAP, Web Services Policy Framework, http://www-106.ibm.com/developerworks/library/ws-polfram/, 10 March (2003)
15. Bachman, F., Bass, L., Buhman, C., Comella-Dorda, S., Long. F., Robert, J., Seacord. R, and Wallnau, K., Volume II: Technical Concepts of Component-Based Software Engineering, Technical Report, May, Software Engineering Institute, Carnegie Mellon University (2003) (CMU/SEI-2000-TR-008)
16. Iribarne, L.; Troya, J.M.; Vallecillo, A., Trading for COTS components in open environments, Euromicro Conference, 2001. Proceedings. 27th, 4–6 Sept (2001) 30–37

17. Ncube, C. and Maiden, N.A., Acquiring COTS software selection requirements, IEEE Software, Vol. 15, No. 2, Mar-Apr (1998) 46–56
18. Seacord, R., Mundie, D., and Boonsiri, S., K-BACEE: A Knowledge-Based Automated Component Ensemble Evaluation Tool, Technical Note, Software Engineering Institute, Carnegie Mellon, December (2000) (CMU/SEI-2000-TN-015)
19. Elfatary, Ahmed, Service Oriented Software: A Negotiation Perspective, PhD thesis, Department of Computation, UMIST, UK (2003)
20. Akao, Yoji, Quality Function Deployment: Integrating Customer Requirements into Product Design, translated by Mazur, G..H. and Japan Business Consultants, Cambridge, MA. (1990)
21. ISO/IEC 9126–1 Information Technology – Software Product Quality Part 1: Quality Model, International Standards Organisation (1998)