Using Web Service Technologies to create an Information Broker

Mark Turner[a], Fujun Zhu[b], Ioannis Kotsiopoulos[c], Michelle Russell[d], David Budgen[a], Keith Bennett[b], Pearl Brereton[a], John Keane[c], Paul Layzell[c] and Michael Rigby[d]

[a]Department of Computer Science, Keele University, Staffordshire, ST5 5BG

[b]Department of Computer Science, University of Durham, Durham DH1 3LE

[c]Department of Computation, UMIST, Manchester M60 1QD

[d]Centre for Health Planning & Management, Keele University, Staffordshire ST5 5BG

m.turner@cs.keele.ac.uk

Abstract

This paper reports on our experiences with using the emerging web service technologies and tools to create a demonstration information broker system as part of our research into information management in a distributed environment. To provide a realistic context we chose to study the use of information in the healthcare domain, and this context sets some challenging parameters and constraints for our research and for the demonstration system. In this paper we both report on the extent to which existing web service technologies have proved to be mature enough to meet these requirements, and also assess their current limitations.

1. Introduction

Web services in various forms are widely seen as an important emerging technology that is still undergoing change and has yet to mature [22]. Within the Pennine Group[1] we have been exploring a longer-term vision over a number of years of how the service concept might be applied to software. Our aim is to use the concept we term *Software as a Service* (SaaS) as the means of overcoming many of the problems that occur with developing, updating and evolving software systems [5, 6, 7, 9]. In Turner *et al.* [22] we examined the principal forms of web service technologies currently available and concluded that these were as yet unable to meet all of the needs of SaaS: in particular, the elements of SaaS that are concerned with the dynamic composition and binding of elementary services at the time of delivery.

However, web service technologies do now enable the composition of less dynamically-changing forms of system, so enabling us to explore some aspects of SaaS. In this paper we describe how we have developed a demonstration broker system for distributed information management using these concepts and, where possible, employing current service technologies. As the basis for this, we have used the healthcare domain, as representing an archetypical, high value, low tolerance example of information management. The IBHIS project (Integration Broker for Heterogeneous Information Sources) is a three-year project funded by the Distributed Information Management programme of the UK's Engineering & Physical Sciences Research Council (EPSRC). Our plan is to use a gradually evolving prototype to explore a range of issues in distributed information management within healthcare, and the system described in this paper is the initial prototype, completed in summer 2003.

Within this context therefore, this paper addresses the research question: "have web service technologies reached sufficient maturity to enable the creation and (rapid) evolution of complex systems?". Indeed, although many papers and web sites advocate the use of different forms of web services, there are very few descriptions in these of other than relatively small and simple demonstrations of their use. IBHIS offers a substantial and complex demonstration, and hence the construction of the initial prototype has enabled us to learn some significant lessons about the use of web services, and about the degree to which these achieve their (and our) aims.

In the rest of this paper we briefly review the state of the art for web service technologies; present the goals of IBHIS and review the system challenges that it presents; describe the form of the first IBHIS prototype; and then use the experiences gained from this to formulate a set of lessons that we have learned, both in terms of using web service technologies, and also in terms of their ability to meet the goals of IBHIS. In the case of the latter, as our research plans are predicated upon the availability of an evolving system to allow us to research different aspects of the domain, we are particularly concerned to determine how well the system as designed is able to support a whole set of possible evolutionary paths.

2. The web services jungle

The first confusing aspect when starting to work with web services is that, as for the component concept, there is no single agreed definition of what the term itself actually *means*. For the purposes of this paper, we use the simple definition proposed by Aissi *et al.* [1]: "software components that use standard Internet technologies to interact with one another dynamically".

[1] The *Pennine Group* consists of software engineering researchers from the University of Durham, Keele University, and UMIST.

Web services are therefore loosely coupled components that can interact independent of platform or language. This independence is achieved through the use of agreed protocols, machine-readable description languages and message formats based upon XML.

Within the web services environment there are numerous XML-based languages and protocols, all offering different functionalities and which, in theory, can be used collaboratively to produce fairly complex systems. These protocols can be conceptually layered on top of one another to form an Open Systems Interconnection style stack, with different protocols offering the functionality required by each layer.

The second problem arises because there is no single agreed definition of this stack, in terms of what layers are required, what protocols should be used, or even which of those available are compatible. For example, IBM [16] and the Web Services Architecture group [8] both have their own versions of such a stack. Turner *et al*. [22] proposed an updated Web services stack framework which included a wider cross section of protocols than most available stacks, including those stemming from the semantic Web initiatives. It is this composite stack that we will refer to throughout this section.

At the simplest level, such a web services stack should consist of messaging, description, and discovery layers. Three XML-based protocols have become de facto standards for providing such functionality. Indeed, these three protocols have become so widespread that the term web services has become synonymous with them:

- Simple Object Access Protocol (SOAP) provides a standard message format for communicating with and invoking Web services;
- Web Services Description Language (WSDL) [24] describes how to access Web services; and
- Universal Description, Discovery and Integration (UDDI) [25] provides a registry that clients can use to discover available services.

These three protocols are adequate for simple service-based development, where locating and binding to a single web service needs to be performed semi-automatically. However, even without considering the more complex non-functional factors associated with SaaS such as electronic contracts, these three protocols do not provide the flexibility required for more complex web service development. A cursory search through the UDDI web interface illustrates that there is an abundance of demonstrations of the (in)famous 'Stock quote' Web service, whereby the name of a company is sent to the service and it returns the company's corresponding stock value. SOAP, WSDL and UDDI are sufficient for providing simple atomic services such as this, but when developing something other than a proof-of-concept technology demonstration, single atomic web services do not provide adequate functionality, and it will be necessary to develop web services that themselves consist of calls to other, lower-level Web services. To this end, other XML-based technologies that deal with the composition of web services have been developed, such as the Business Process Execution Language for Web Services (BPEL4WS) [2] and the Business Process Modelling Language (BPML). Technologies are also needed to describe the order to call the methods—or operations in WSDL terminology—of the services and to deal with the issue of long-running transactions that web services composition introduces. Initiatives such as the Web Services Conversation Language (WSCL) [4], and the WS-Transaction [11] and Business Transaction Protocol [18], respectively aim to address these issues.

While there are protocols available that can deal with the majority of technical issues concerning web services development, the question of how successfully an application can be built with them is still open. In the rest of this paper we aim to outline our experience of using a sub-set of these protocols to build an information broker, and in the next section we describe the goals for this.

3. IBHIS: a healthcare information broker

The widespread development of computer networking has made it possible to transfer large quantities of information between users and sites using well-established protocols. However, this vision (for example, as espoused in the seminal US Institute of Medicine study [14]) is short of analysis of practical solutions, and subsequent national policies are still heavily dependent upon concepts of large organisational enterprise systems whose shortcomings have been noted in the commercial sector [19]. Moreover, the *integration* of that information to meet enterprise or user needs presents a much more difficult problem, 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 *broker* (such as a travel agent), who is able to meld information on demand. In the IBHIS project we are seeking to create an *information broker service* that will support the reliable integration of information that is held in heterogeneous forms, and managed by autonomous agencies, using the healthcare domain as our exemplar.

While the detailed funding and organisation of healthcare may vary substantially between different countries, the essential properties of information exchange and management do not differ. Hence, although our project is based upon the structures that currently apply to the U.K., its applicability is worldwide.

In the healthcare domain, the National Health Service of the U.K. is, in systems terms, a classic large dysfunctional organisation. Yet within this, autonomous

agents are expected to deliver seamless care to the patient. Primary care practices, hospitals, community trusts, mental health trusts, and health authorities are all independent organisations, each with its own information system. In addition, the funding of social services is organised on a totally different national basis.

Within this very wide, varied, and continually changing context, we have chosen to employ a set of case studies as the basis for our information models. These case studies have been drawn up with the aid of our collaborators in Solihull Primary Care Trust, which has acted as the field site for the IBHIS project.

The six case studies (not all of which have been modelled in equal depth) were selected as representing a mix of health and social care needs with complex information flows, and are titled as:

- Disabled children with complex needs
- Mental health of 'looked-after' children
- Child protection
- Single assessment process
- Intermediate care
- Mental health

While we do not have space to expand on these here, we should also note that these headings represent local implementation of issues that appear on national agendas and hence which are fairly well-defined. They also represent a complex and rich set of information flows.

In technical terms, these require the IBHIS broker to do more than simply function as a federated database (itself a challenging task as explained later). Information that is held by separate agencies may well be indexed using different keys (for example, the Health Service uses a Patient Record Number fairly extensively, whereas Social Services employ quite different keys). A further set of issues arise concerning custodianship, authentication and ethical control of information—while the goal of IBHIS is to demonstrate what is *technically* possible, even if this is not permissible under current regulatory structures, this does not remove any overall ethical considerations concerning access to patient information. There are also issues concerning the access to information by *role*, which in turn leads to the need to be able to provide for the possibility that individuals may take on different roles at specific times, and also that authority may be transferred by mandate when necessary.

The next section describes the architecture of the fully service-oriented broker and identifies some of the mechanisms we are employing to address issues such as those above. By necessity, this section provides only a brief summary of what is a very complex problem domain and of the processes by which we are seeking to model its needs in order to create the domain models needed for IBHIS.

4. IBHIS Service-oriented architecture

In order to provide the end-user with a unified view of data on demand from autonomous heterogeneous data sources, we propose a *Service-Oriented Data Integration Architecture (SODIA)* as illustrated in Figure 1, which is based on the concepts of SaaS and its use of dynamic (late) binding. It is this architecture that we plan to employ within the final prototype IBHIS broker.

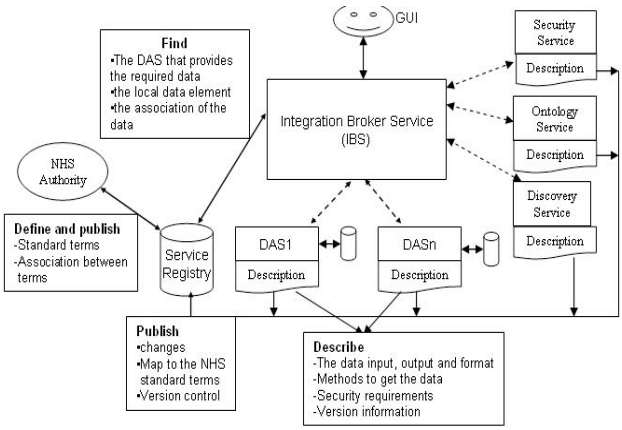

Figure 1. Service-Oriented Data Integration Architecture

An explanation of the architecture is as follows:

1) We assume that all the elements such as business content (vocabulary), constants (measure, postcode), data types and intent are defined by an authority within the Health and Social Care domain, and described using an ontology-based approach. This ensures that they can be interpreted correctly by the service requesters and providers.
2) Service providers (including data service providers) implement services and describe them using service description languages such as WSDL and DAML-S [3].
3) Service providers publish the service description file into a service registry, such as UDDI, and map the local data elements to the standard terms defined by the Health and Social Care authority.
4) Services are located using the *service registry* and the service implementation is invoked via SOAP. When an end user wants to get integrated data, he/she looks for the UDDI registry and finds a suitable *Integration Broker Service (IBS)* which satisfies his or her requirements. When the IBS is invoked, it dynamically finds the sub-services (data and functional services) and binds to them.
5) The *discovery service* is used to discover and bind to service implementations at run time. When there is more than one service providing the same function, it can be used to choose one service based on the user's requirements.
6) An *ontology service* is used to provide semantic transformation for different data items.
7) The *security service* is used to authenticate the user and control the data items that a given user can access.

8) The *Integration Broker Service* (IBS) itself is a composed service, which integrates different data access services and functional services such as the *Discovery Service* (DS) and provides the end-user with an integrated uniform view of the data. The backbone is the *workflow service*, which manages the business logic, i.e. how different data services work together to provide integrated data. The IBS could also compose a negotiation service to perform negotiation with data sources.
9) A *Data access service (DAS)* is a variation on a typical (Web) service as it is more data intensive and designed specifically to expose data as a service. Data service providers implement a DAS, which may query multiple, heterogeneous data sources. Alternatively, different DASs may query the same data source but produce different data outputs.
10) When data service providers publish a DAS description file into the registry, they also publish the associated DAS metadata and ontology.
11) For reasons of change management, DAS providers may provide users with different versions of their DAS. The service consumer can decide which one better meets their requirements, or choose to bind to the latest version by default.
12) Unlike the functional service, DASs are stateful and service transaction mechanisms may be employed (Section 2).
13) Because services could be recursive, the IBS could also be composed with other services such as a security service, or a transformation service to provide a more value-added service.
14) There is no integrated schema; data is integrated on the fly.

In the next section we describe the form of the initial prototype broker as completed in the summer of 2003, including details of the technologies used and the methods employed to expose a number of sample healthcare data sources as services.

5. The initial prototype

The aim of the IBHIS project is to develop a series of prototype information brokers of increasing complexity. Each subsequent version of the prototype will realise more of the ongoing SaaS research until the final version is based around the SODIA architecture (Figure 1).

The following section describes the implementation of the first prototype, which incorporates an initial sub-set of the service-oriented elements.

5.1. System architecture

The underlying architecture of the first prototype is based around a fusion of the available web service protocols with the concepts of a Federated Database System (FDBS). A detailed breakdown of the IBHIS architecture in relation to FDBS can be found in Kotsiopoulos *et al*. (2003) [15].

By its very nature, a major part of the IBHIS project is concerned with the seamless accessing and integration of many forms of complex data and exposing it as a service. As many service-based projects are concerned with exposing *software* as a set of services and not *data,* one of our research questions was how to realise traditional database structures within a service-oriented architecture, especially given the limitations of the available protocols. It was with this in mind that we chose to employ a traditionally database-oriented concept –that of the federated schema—but realise it within a service-based implementation. We use web services to enable the provision of data on demand whilst keeping the underlying data sources autonomous. This architecture allows us to provide *data* as a service (DaaS) as oppose to *software*.

5.2. Service elements within IBHIS

Figure 2 illustrates how, conceptually, the first IBHIS prototype is made up of several component 'services'.

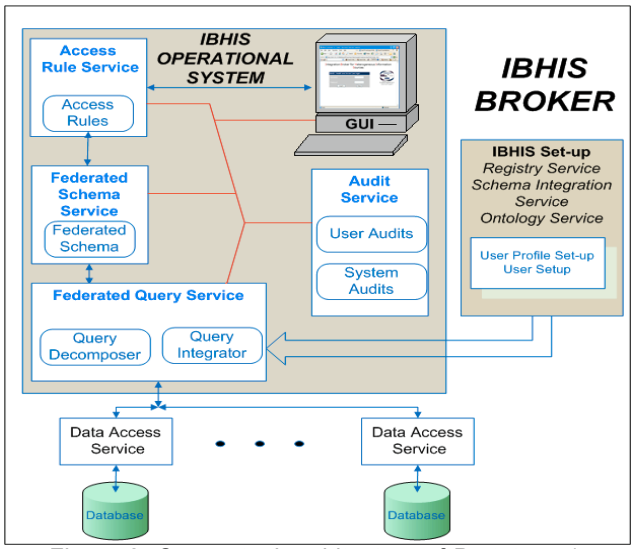

Figure 2. Conceptual architecture of Prototype 1

Access Rules Service (ARS)

The ARS is responsible for the initial user authentication and subsequent data access authorisation. Within the Health and Social Care domain there are a number of clearly defined roles each with their own set of generalised data access rights. Much research has been conducted into applying Role Based Access Control (RBAC) within the Health domain [20], with the conclusion that RBAC alone is too inflexible for health care applications. Therefore, a more complex set of access rules were developed in conjunction with Solihull Primary Care Trust, incorporating individual user rights with roles. This prototype incorporates an initial set of

these, which will gradually be expanded and implemented fully in subsequent prototypes

Federated Schema Service (FSS)

The FSS maintains the Federated Schema and all of the mappings between the local Export Schemas and the global Federated Schema. The FSS is consulted by the Federated Query Service during query decomposition.

The Federated Schema and the corresponding mappings to the Export Schemas are bound during broker initialisation.

Federated Query Service (FQS)

The FQS contains two sub-modules:

1. The *Query Decomposer* decomposes the Federated Query into a set of local queries; this is done in consultation with the FSS.
2. The *Query Integrator* receives the set of local results from the Data Access Services and integrates them into a Federated Record.

The FQS sends the Federated Query and the Federated Record to the Audit Service.

Audit Service (AS)

The AS will contain two sub-modules which will keep track of every action of IBHIS that needs to be recreated or audited in the future, including user (login date and time, IP address and sequence of queries executed) and system audit details (data source registration and user setup).

Data Access Services (DAS)

In the first prototype, DASs perform conceptually the same function as was described in section 4. The service description of a DAS must allow the consumer to discover:

- The data input, output 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 service providers then publish the description file into the service registry so that a consuming service, in our case the FQS, may discover them. Essentially, the DAS is used to provide a transparent, unified way to access distributed, autonomous heterogeneous data sources.

Table 1, below, illustrates how the conceptual services implemented in the first prototype can be mapped onto the SODIA architecture (Figure 1).

SODIA	First Prototype	Notes
IBS (integration broker service)	FQS	The FQS is used for integrating data only, but IBS is a broker service that will call other services if necessary.
Data Access Service (DAS)	DAS	Similar, but in prototype 1 we do not use semantic descriptions.
Security Service	ARS	ARS is imposed at the beginning of the session rather than on a per-use basis of the DAS.
Ontology Service	FSS	In prototype 1, it is not an individual service, it is part of the FSS
Discovery Service	FSS	Included within the FSS
Service registry	FSS	Implemented as a mappings database in prototype 1

Table 1. Mapping prototype 1 to the SODIA

5.3. Implementation

The following section details the technologies chosen along with the reasons behind their choices.

Development and Deployment Tools

Whilst web services are language independent, there are two main choices for programming platform [23]: Microsoft's multi-language .NET or Sun's J2EE framework. An evaluation of both technologies was performed through the use of both a SWOT and feature analysis with the result that the Java based J2EE appeared more suitable for IBHIS. This was largely due to the support for multiple platforms.

Once the programming language was decided upon, a set of application servers and Integrated Development Environments (IDE) were needed. The choice of J2EE as a programming environment to some extent dictated the choices as we required tools that were compatible with Java. Five available servers and IDEs were evaluated, with IBM's Websphere suite of tools appearing to fulfil our requirements. Specifically, Websphere Studio Application Developer version 5 was chosen as the IDE along with Websphere Application Server version 5. We aimed to use a consistent development environment throughout the three sites to ensure that any problems encountered were not due to version or tool incompatibility. The use of a number of servers allowed us to deploy the individual services at different sites in order to build a truly distributed system.

Finally, we needed to provide a set of heterogeneous, distributed data sources to provide a context within which to test and demonstrate IBHIS with respect to complex data integration. Three DBMS platforms were chosen—MySQL, IBM DB2, and Oracle—which were deployed on servers on site within each of the universities. The content was designed to simulate typical healthcare agencies based upon the case studies of

section 3, such as General Practitioners and Social Services, with all of the semantic and syntactical heterogeneity that exists in the domain.

IBHIS Service elements

Figure 3 illustrates a schematic of the first prototype. As can be seen, not all of the conceptual services modelled in Figure 2 were actually implemented at this stage, particularly the audit service. This has been left to a later increment of the prototype:

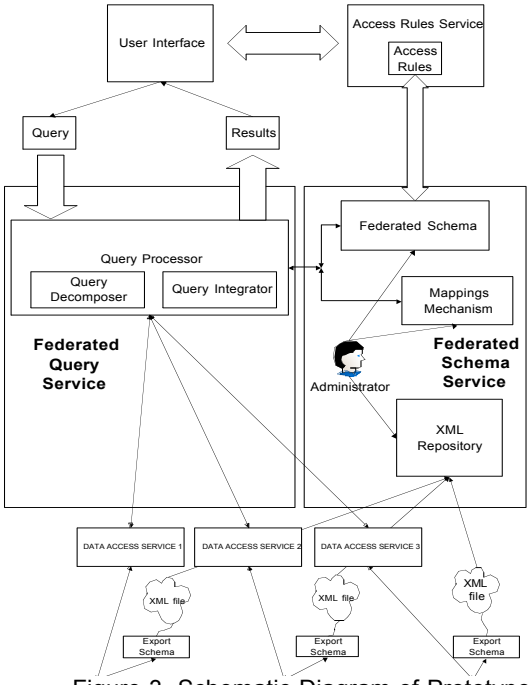

Figure 3. Schematic Diagram of Prototype 1

ARS

For the first prototype we used the ARS to restrict access to the federated schema (itself implemented as an XML Schema document) according to a set of rules, and to create a 'view' of the schema for that session.

A number of possible policy language specifications were evaluated, including the XML Access Control Language (XACL) [17] and XACML [13]. We chose to use XACL as it offered fine-grained access control to XML elements. We believed XACML, whilst a more recent specification, was not appropriate for the type of document control required.

FSS

The FSS is mainly used at design time to create the federated schema and the corresponding mappings. The export schemas of the component databases are retrieved in XML format and stored in a database. The system administrator, along with a domain expert, resolves any semantic differences and creates the federated schema. In our model, the mappings between the export schemas and the federated schema are stored in a relational database. The database is wrapped as a web service and the appropriate mappings are exposed as methods.

FQS

The query that the FQS receives consists of a list of attributes, a search criterion and its value. The query is analysed by a query decomposition algorithm and a set of methods are called from the underlying data access services. The results from the DASs are subsequently integrated into a single record according to the federated schema. The query decomposition algorithm relies on the mappings mechanism to resolve any semantic problems such as synonyms and homonyms.

DAS

In a real situation, service providers may choose to implement their DASs differently, so to simulate this we used two different programming designs. Two DASs were programmed using JDBC to access their databases directly, while the two remaining DASs used J2EE. Entity beans were used to access the databases, with session beans accessing the entity beans. The session bean was then wrapped as a web service to form the DAS.

5.4. Web service protocols

Each of the service elements were implemented as described and built as one or more distributed web services. To achieve this it was necessary to use a number of different web services protocols to invoke, describe and bind to each service. However, as outlined in section 2, there is an extensive choice of protocols that can be used.

In terms of messaging and invoking services, SOAP is the only real choice. Within SOAP itself there are two choices for messaging style: Remote Procedure Call (RPC) or Document style messaging. RPC style messaging is based around a traditional request/response interaction, with proxies being used at the client side of the interaction to deal with the message construction. By comparison, Document style messaging is asynchronous and requires the client to build the SOAP messages themselves, using an agreed XML syntax.

There are a number of advantages and disadvantages to both styles of messaging but for the first prototype we chose to use RPC with SOAP data encoding, largely because of the static nature of the service interfaces and also because of the potential performance benefits offered. The latter is especially important when building a system that could be used in a 'real-time' healthcare environment

Service based registries, such as UDDI, are used to allow clients to dynamically discover services. Within IBHIS, some form of registry is needed to enable the FQS to discover which DASs can provide the appropriate data to solve a user's query. This discovery needs to be performed at run-time to enable the FQS to retrieve the location and interface details of appropriate DASs. However, we believed that current service-based registry implementations, and in particular service description

languages, were lacking in certain areas required to dynamically locate and bind to DAS services (see section 6) and as a result chose not to employ a standard service registry. Our solution was to use a 'registry' database which included mappings to DAS interfaces that could provide each attribute in the federated schema.

As illustrated in Turner *et al*. [22], there are numerous available languages with which to describe web services, all offering slightly different content and with varying support within the community. The choice of which to use depends greatly upon the project and the degree to which factors such as dynamic binding, composition, and negotiation are required. Whilst many of these features will become important in future prototypes (see section 6), most were of only minimal importance to the first prototype. Composition, for example, will be important as we move towards the architecture described in section 4, as services will be discovered in a registry and dynamically composed. In this situation, composition languages such as BPEL4WS will be implemented, whilst in the first prototype services are statically bound at design time, making composition languages unnecessary.

We chose to use the de-facto standard description language WSDL for the first prototype as we believed it provided adequate descriptions for our initial requirements. Also, the fact that it was supported by our chosen development tools enabled us to benefit from rapid development.

6. Discussion and observations

Overall, the development of the first prototype was successful, in that it was completed on time without any major technological changes, and enabled the integration of distributed, heterogeneous data based upon healthcare case studies. It is currently in the process of being reviewed by experts within the healthcare domain.

However, the development was not without problems and as explained in section 5 it was necessary to employ a number of alternative temporary solutions in order to complete the first prototype. Some of these arose from our focus on exposing traditional database structures rather than software, thus revealing limitations in the current web services protocols by deploying them in an unfamiliar environment. It is also beneficial to examine how we believe the technologies employed will allow us to evolve the first prototype towards the architecture outlined in section 4. In doing so, we will be able to assess if current web services technologies are mature enough to meet our needs. Firstly, we examine our choice of architecture.

6.1. Architecture

As explained above, due to our focus on data and the limitations of service protocols (see section 6.3), we chose to base the architecture of the first prototype around a hybrid form using web services together with a Federated Database System. This architecture is an example of a tightly coupled FDBS with multiple federated schemas. The advantage of such an architecture is that it allows the creation of multiple integrated schemas according to the system requirements. This was realised in IBHIS by dynamically producing different views of the federated schema according to a set of access rules. The federated architecture proved to fit relatively well with the concept of *data as a service* as long as the schemas of the underlying data sources did not change frequently and the number of subscribed data sources remained relatively small. Manual schema integration is a time consuming and error prone procedure which makes frequent changes in the underlying data schemas difficult to manage. Given that the federated schema is built at design time, frequent changes would result in serious functionality problems.

As a result, one of our research objectives is to design a more flexible version of the proposed architecture, and within this to seek to use the service model as described in section 4 to overcome this problem. Along with this goes the notion of a 'meta-IBHIS', in which a hierarchy of communicating IBHIS brokers, each themselves exposed as services, cooperate to connect islands of data.

6.2. Web services protocols

A number of different web services protocols and languages were employed to create the first prototype. In this section we progress beyond the architectural discussions to discuss our protocol choices and assess the extent to which they will facilitate the system's evolution.

At the most basic level, our deployment of SOAP based services proved to be successful. Our choice of RPC web services meant that local client proxies were used, giving performance benefits and a shallower learning curve, as the proxies dealt with the message creation and data type encoding.

However, a number of problems were encountered as a result of our choices. One problem was related to the format in which data could be sent between services. In our example, the interface needs to send a query to the FQS. If Document style messaging had been used, the query could be dynamically constructed and sent as a single XML document, using pre-specified element names and XML Schema data typing. However, when using RPC web services the developer is tied to using data types that are consistent with section 5 of the SOAP 1.1 specification [10]. Also, as the SOAP message is not under the developer's control there is much less flexibility in terms of element names, as they are tied to the names of methods and parameters. The result was that the services were forced to return the results as array data types to fit in with Java's method based structure. Whilst a concern, this in itself was easily solved. However, arrays could not be sent as parameters due to problems with SOAP data encoding. Therefore, instead of sending an array as a query parameter it was necessary to send it

as a concatenated string, requiring the receiving service to perform unnecessary work in decomposing it.

Also, a great deal of time was spent during development recreating the local proxies and changing the requesting service every time the service interfaces changed. The project is addressing a rapidly changing domain and transient requirements, therefore service interfaces are not only going to change during early development but are almost certainly going to change regularly throughout the project's lifetime. The use of RPC messaging produces a tightly coupled system, and therefore it is unlikely that RPC will provide sufficient support for the planned evolution of the prototype [12]. RPC was only able to support the development of the first prototype because we were able to specify the form of the interfaces in advance. In the future this may not be possible. As a result, we will move towards the use of Document messaging and literal data encoding. However, the problems encountered were largely due to our particular usage of SOAP, and we believe that as a protocol it is able to provide everything necessary for us to progress towards our service-oriented architecture.

It is difficult to examine our actual experience with service-oriented discovery as we chose to employ an alternative registry implementation rather than use UDDI. Obviously, this approach was tailored more towards our hybrid architecture than a pure service-oriented route and in that respect it worked successfully, enabling the system to locate and bind to DASs at run-time. However, the problem is that it requires an extensive prior knowledge of the data sources and the DAS interfaces. If these were unknown, the system would fail. Also, the frequent addition of new data sources or changes to the existing schemas will quickly render this solution unmanageable.

Hence, we are currently investigating adding a service-oriented registry to the prototype. This would provide a more flexible design by allowing the FQS to dynamically discover and bind to any number of DASs at run-time, without requiring the FQS designer to have extensive prior knowledge. However, the major problem with implementing a service registry and the reason why we decided to use an alternative solution is because we are exposing *data* as a service and not software, which requires an extremely detailed service description which the majority of available languages lack.

WSDL was the sole language used to describe each web service within the prototype. It was created at design time by the chosen IDE, and used to provide the interface with which to create the client side proxies. WSDL was able to provide adequate descriptions to allow the development of the first prototype, but we believe it will not provide the descriptions necessary to implement our SODIA architecture (Figure 1). If a registry structure is implemented then the FQS will search the registry and hence the service description files with the aim of dynamically discovering DASs that can solve a query. However, in order to achieve this, the registry must allow for reasonably semantic, flexible searching and the service descriptions must be rich enough to describe the service in both a technical sense and also in terms of the data that it can provide. WSDL only describes a service in terms of its data types, methods and parameters, message format, transport protocol, and end-point uniform resource identifier (URI). Whilst using literal data encoding within the WSDL file and not SOAP encoding would allow a consumer to discover what data the service could provide, it does not describe other significant factors required when accessing autonomous data sources in a service-oriented environment. For example, WSDL lacks descriptions of the service's security requirements, versioning details, quality of service and other non-functional descriptions. Security is particularly important as without a federated schema, the ARS will become more service-oriented, granting or denying permissions on a per-use basis in collaboration with the DAS's own security measures. Also, without reference to an ontology of terms and semantics, it is difficult for consumers to automatically discover the meanings behind services. Therefore, it is unlikely that WSDL would be suitable for our needs, at least not without significant extensions.

Ontology-based description languages such as DAML-S offer richer service descriptions, and so it is feasible that it would be adequate for a service-based architecture. However, DAML-S has not yet reached a final release, meaning that the finer details of the language are still in flux. Also, the fact that tools do not offer widespread support for the language limits current possibilities for rapid application development. Even if the language reaches a formal release, there will still be problems with the currently available registries in that they are largely keyword based, which limits the possibility for truly dynamic searching.

6.3. Service-oriented development

We and others believe that service-oriented development offers considerable benefits over traditional software development paradigms [5, 6, 7, 9, 21]. Such benefits include rapid software evolution and updating, dynamic discovery, configuration and binding to services on a per-use basis, and the dynamic creation of new services composed from other services. Whilst web services do not currently allow implementation of all of these features [22], they are the closest available technology. Despite this, there appears to be little documented evidence of developer's experience of web service development and the benefits it can bring. In this section therefore, we have reviewed some of our experiences with web services development in general.

It is claimed that web services offer complete platform, language and implementation independence. Despite small incompatibilities between minor IDE versions, this proved to be true in our case. The DASs in particular were programmed using different underlying concepts and deployed at distributed sites on heterogeneous platforms, accessing a number of different DBMSs. This is all transparent to the consuming service –the FQS—because of SOAP's unified XML data

format. The separation of the service description and implementation also means that the implementation of the service could change without changing the code of the consuming services. This independence also eased the process of building the system independently. It allowed the developers to code and deploy the individual web services within their local application servers and be confident of 'plug and play' interaction between them.

As regards updating, in our architecture the FQS changed fairly regularly. However, the developer was able to deploy new versions in their local application server without affecting the system. The other services were still able to communicate with new versions of the FQS without changing the code or restarting. Therefore, this feature combined with the independence provided by XML messaging, should allow the evolution of the system towards a service-oriented architecture much more rapidly and with less disruption than traditional software paradigms.

6.4. Development and Deployment Tools

The development team benefited a great deal from using IBM's Websphere platform. The version of the Websphere development platform used provided all the facilities needed to implement, test and deploy web services. The web services were deployed in Websphere Application Server. The process of creating and deploying web services was carried out without any major problems, and the IDE offered a number of time-saving features in the form of automatic WSDL and XML creation. An added bonus was the Concurrent Versions System which is an open source plug-in for Websphere providing version control for distributed development.

However, it has to be noted that a number of problems arose during the first months of the project with the early versions of Websphere Application Developer v5, but these seem to have been considerably reduced in the latest version. However, despite the successful improvements, it was always felt that these tools have problems with backward compatibility.

Overall, Websphere appears to have matured enough to support the seamless development of web services applications.

7. Conclusions

The primary aim of this paper was to assess the extent to which web services could enable the successful development and deployment of a complex, data-oriented system and whether they are sufficiently mature to allow the rapid evolution of this system towards a completely service-oriented architecture.

As Section 4 illustrates, we are aiming to develop an extremely complex service-oriented broker, and in Section 5 we described the implementation of the first prototype, which is the first step to realising a data-oriented problem domain with a service-oriented design. Our choice of IBM's Websphere set of tools proved to be successful, offering a consistent development and deployment environment. The combination of SOAP, WSDL and our own 'registry' implementation was effective in handling distributed deployment across heterogeneous platforms and allowed for relatively transparent evolution of the component services. It was discovered that the federated schema architecture was too statically bound to the data sources and required a great deal of manual editing at design time to accommodate change. However, this architecture was only a temporary measure to allow us to tackle the problems of heterogeneous data integration, and to discover any potential web services problems before building the service-oriented architecture.

The problems we have identified arose from a combination of our particular protocol choices, and problems with using the Web protocols in general to access traditional database structures. The former were easily solved, whilst the latter are more difficult and provide issues for further research. For example, we have identified problems with using WSDL for our needs, in that it does not offer the type of flexible, semantic, non-functional descriptions we require when dynamically locating Data Access Services. Confusion can arise in the meanings of service and parameter names without semantic descriptions, and without non-functional descriptions essential features such as quality and even security are neglected. There are other languages—see section 2—but these either lack the support of tools or registries or are currently unfinished. Therefore, it is likely that we will need to extend one of the available languages to fit our needs.

Registries also offer problems for flexible searching using methods other than keywords, for example ontology based searching or searching by function. Also, it is unlikely that the Integrated Broker Service (IBS) can always find the exact DAS(s) it needs to meet the user's requirements, so therefore the matchmaking mechanism needs to be investigated to ensure that the end user can still get some of their results. Another challenge is how the IBS can evolve with the changes made by individual, autonomous DAS owners.

Overall, our experience with web service technologies when developing the first IBHIS prototype and planning the system's evolution has been positive and has offered many benefits over other programming and deployment paradigms. However, we believe that there are still some areas where web services are not yet mature enough to meet the requirements of a complex system such as IBHIS, particularly as regards service description languages.

Acknowledgements

We thank the members of the Pennine Group for their continuing research into Software as a Service and for contributing to the ideas in this paper. We also thank the

members of the IBHIS project, including our collaborators in Solihull, for their useful discussions during the development of the first prototype and for their insight into the needs of the Healthcare domain. This work was funded by an award from the Distributed Information Management programme of the Engineering and Physical Sciences Research Council.

References

[1] S. Aissi, P. Malu and Srinivasan. K, "E-Business Process Modeling: The Next Big Step", *IEEE Computer*, **35** (5), 2002, pp 55-62

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, I. Trickovic and S. Weerawarana, "Specification: Business Process Execution Language for Web Services Version 1.1",2003; http://www-106.ibm.com/developerworks/library/ws-bpel/

[3] A. Ankolekar *et al*., "DAML-S: Web Services Description for the Semantic Web," in *Proc. 1st Int'l Semantic Web Conf.* (ISWC 02), Springer-Verlag Heidelberg, 2002, pp. 348-363.

[4] A. Banerji, C. Bartolini, D. Beringer, V. Chopella, K Govindarajan, A. Karp, H. Kuno, M. Lemon, G. Pogossiants, S. Sharma and S. Williams, "Web Services Conversation Language (WSCL) 1.0" 2002; http://www.w3.org/TR/wscl10/

[5] K. H. Bennett, P.J. Layzell, D. Budgen, P. Brereton, L.Macaulay and M. Munro, "Service-Based Software: The Future for Flexible Software", in *Proceedings of 7th Asia-Pacific Software Engineering Conference*, IEEE Computer Society Press, December 2000, pp214-221.

[6] K.H. Bennett, J. Xu, M. Munro, Z. Hong, P.J. Layzell, N.E.Gold, D. Budgen and O.P. Brereton, "An Architectural Model for Service-Based Flexible Software", in *Proceedings of COMPSAC'01*, IEEE Computer Society Press, October 2001, pp137-142.

[7] K.H. Bennett, M. Munro, N.E. Gold, P.J. Layzell, D. Budgen and O.P. Brereton, "An Architectural Model for Service-Based Software with Ultra-Rapid Evolution", in *Proceedings of ICSM'01*, IEEE Computer Society Press, November 2001, pp292-300.

[8] D. Booth *et al*., "Web Services Architecture: W3C Working Draft," 14 May 2003; www.w3.org/TR/ws-arch/

[9] O.P. Brereton, D.Budgen, K.H. Bennett, M. Munro, P.J.Layzell, L. Macaulay, D. Griffiths and C. Stannett, "The Future of Software", *Comm. ACM*, **42**(12), December 1999, pp78-84.

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

[11] F. Cabrera, G. Copeland, B. Cox, T. Freund, J. Klein, T. Storey and S. Thatte, "Specification: Web Services Transaction (WS-Transaction" 2002; http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/

[12] T. Ewald, "The argument against SOAP Encoding", 2002; http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsoap/html/argsoape.asp

[13] Godik. S *et al.,* "OASIS eXtensible Access Control Markup Language (XACML): Committee Specification 1.0 (Revision 1)", December 2002; http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-01.pdf

[14] Institute of Medicine, "The Computer-Based Patient Record – An essential Technology for Health Care (R.S Dick and E.B Steen, eds.)", 1991; National Academy Press, Washington DC

[15] I. Kotsiopoulos, J. Keane, M. Turner, P.J. Layzell and F. Zhu, "IBHIS: Integration Broker for Heterogeneous Information Sources", to appear in *Proceedings of COMPSAC'04*, IEEE Computer Society Press.

[16] H. Kreger, "Web Services Conceptual Architecture (WSCA 1.0)," 2001; www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf

[17] M. Kudo and S. Hada, "XML Document Security based on Provisional Authorization", in *Proceedings of 7th ACM Conference on Computer and Communication Security*, November 2000, pp87-96.

[18] OASIS Business Transactions Technical Committee, "Business Transaction Protocol. An OASIS Committee Specification. Version 1.0", 2002; http://www.oasis-open.org/committees/business-transactions/documents/specification/2002-06-03.BTP_cttee_spec_1.0.pdf

[19] E. Oz, "When Professional Standards are Lax: The CONFIRM Failure and its lessons", *CACM*, **37**(10), October 1994, pp29-36

[20] J. Poole, J. Barkley, K. Brady, A. Cincotta, and W. Salamon, "Distributed Communication Methods and Role-Based Access Control for Use in Health Care Applications," *Second Annual CHIN Summit* 1995.

[21] C. Szyperski, "Component Technology – What, Where, and How?", in *Proceedings of 25TH International Conference on Software Engineering*, IEEE Computer Society Press, May 2003, pp684-693

[22] M. Turner, D. Budgen and O.P. Brereton, "Turning Software into a Service", *IEEE Computer*, **36**(10) October 2003, pp??.

[23] C. Vawter and E. Roman, "J2EE vs Microsoft .NET. A comparison of building XML-based web services", 2001; http://www.theserverside.com/resources/articles/J2EE-vs-DOTNET/article.html

[24] Web services description language (WSDL) 1.1. http://www.w3.org/TR/wsdl

[25] UDDI version 2.0 API specification. http://www.uddi.org