Giving legs to the legacy — Web Services integration within the enterprise

J Calladine

Behind the hype of Web Services (WS) is the reality of a sound and functional set of technologies for integration. This paper describes how the pragmatic use of WS technology has resulted in better, cheaper and faster integration for BT in its core operational support systems (OSS) system interfaces. BT has embarked upon a programme of enabling key components and middleware layers in its customer handling stack to leverage the benefits of Web Services in addressing the challenge of real-time heterogeneous interworking. It is believed that the WS-enabling of these legacy systems has helped add years to their productive use and protected the huge investment already made in these systems. As well as giving 'legs' to such legacy systems, Web Services support the drive to implement a service-oriented architecture inside BT and this paper investigates the ways in which this is being achieved. In order to exploit Web Services on BT's mission-critical platforms we have had to understand and work with WS issues on a wide range of platforms, proving the interoperability, performance and security of the new technologies in large-scale enterprise integration projects. Integration has been the first 'killer application' for Web Services and this paper will describe some of the business areas that have benefited from the cleaner integration that Web Services offer.

1. Introduction

Web Services technology represents the latest step on an evolutionary path for the integration of BT's major systems using a common middleware infrastructure. This section briefly describes how Web Services integration has come to be the current strategy direction for enterprise integration.

An earlier paper [1] described the various ways in which BT's highly heterogeneous OSS estate had been integrated using a selection of middleware infrastructure products. This work has continued, leading to even greater exploitation of these large mission-critical systems via programmatic interfaces within the extended enterprise.

With some of these major 'legacy' systems approaching 15—20 years of operational use, the range of middleware used to access them has itself grown diverse (see Fig 1). At the time of writing, BT's legacy systems are accessed via a range of protocols and mid-tier technologies including:

- DCE/RPC,
- CORBA/IIOP,
- J2EE/RMI/HTTP,
- SNA LU2/LU6.2,

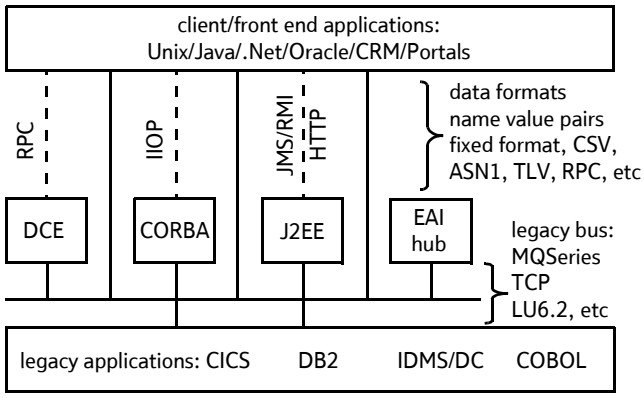

Fig 1 The range of access technologies in use.

- 'raw' TCP/IP sockets,
- MQSeries.

Apart from the native data representation of RPC-based technologies such as DCE/CORBA, data has been represented in a number of ways including:

- name value pairs,
- ASN1 encoding,
- comma-separated variables,
- tag length value,
- fixed format 'copybook'.

The ultimate aim of each generation of integration technologies has been to provide a near-universal access solution for all clients and this has largely failed due to the highly heterogeneous nature of BT's systems. Often the mid-tier or middleware solution used was tied to a particular language or set of languages which themselves were not universally available on all client platforms. Sometimes the vendors would only support a restricted range of platforms (it was quite common to only support three or four major Unix platforms) and sometimes the solution could not be cost justified for deployment on thousands of desktop systems. Even where platforms and languages were supported, the use of some solutions involved an invasive tight coupling and embedding of stub code in each client that presented an 'unnatural' style of interface to the client developer.

Ideally we would have a common technology that was not tied to a particular set of languages or platforms and the invocation of which could be performed naturally in each environment present in the BT architecture. Such a technology would allow us to wrap our existing services as well as deploy new ones and facilitate a migration of services on to the new technology in a way that was acceptable to the base of client systems already integrated.

In 2000 we made a strategy decision that all new services would be made available over HTTP and MQSeries and offer an XML data representation. This was irrespective of any other 'native' interface that the service might offer, e.g. RMI or IIOP. The ambition was to provide a lowest common denominator access protocol that would serve all platforms in BT and offer both a synchronous and asynchronous mode of inter-working. This strategy was implemented on new systems and at the same time we began to re-engineer our middleware infrastructure to enable our existing legacy systems to be driven in this way.

From this position we then moved to adopt the emerging SOAP standard to add some formal rigour to the XML documents that were being exchanged and we tracked the emergence of the WSDL standard and its growing support across the industry. It was this overwhelming industry support that led us to become an early adopter of the Web Services model in 2001 to facilitate improved integration with a wide range of client systems.

2. Benefits of Web Services for enterprise integration

As mentioned above, a key factor in adopting Web Services was the wide industry support that Web Services attracted. Web Services tool-kits were becoming available for every imaginable platform and language combination. Key suppliers to the BT infrastructure (such as BEA, IBM, Microsoft and Oracle) were moving to support the standard and so there was the promise that our existing platforms would be WS-enabled natively. For all other systems there appeared to be solutions available either from open source packages (Apache, gSOAP, SOAP:Lite, etc) or from specialist suppliers (Cape Clear, Systinet, The Mind Electric, etc).

This industry support meant that we were confident that, if we provided a Web Services interface to our major systems, anyone would be able to invoke these services using a native interface style. This could be done without the tight coupling of certain vendors' stacks and in a cost-effective fashion using one of the many tool-kits available on the market.

One of the advantages of the Web Services model is that developers do not build a 'Web Service'. Instead they code a Java bean or a C++ module or a script that in turn gets rendered as a Web Service — the complexities of the XML, SOAP and WSDL are hidden from them. Likewise developers of client systems do not know they are invoking a Web Service but instead see a native function call in their own environment.

We also found that the XML language and the schema component of WSDL were rich enough to represent and describe the structure of any of our existing interfaces[1]. Furthermore, as the conversion of existing services into Web Services in the middleware layer (i.e. not rewriting the service itself) has proved relatively straightforward, a large number of Web Services can be made available at little cost.

In effect, what we have been able to achieve with the use of Web Services technology is a consistency of interface at both the transport and data levels. We say that we are 'homogenising' the application's integration space[2]. The result of this we believe is that we are extending the useful life of these core systems by bringing them up to date and enabling better, cheaper and faster integration with them and reducing time to market for new products that exploit these systems.

Web Services enabling of these legacy systems is a cost-effective way of giving them the 'legs' they need to continue as an available and attractive source of the mission-critical data and functions that they host. We have found that the provision of a WS interface allows us to communicate directly with these legacy

[1] It is not, however, comprehensive enough to easily describe the data content of the interface. The strong typing of data as exhibited by, say, DCE is not easily possible and this level of validation within the integration middleware has been sacrificed.

[2] Websters dictionary definition: 'To render uniform and consistent.'

applications from front-end systems avoiding the need for costly mid-tier developments.

3. Enabling the infrastructure

A large amount of work on many systems has been done in BT but here we shall focus on some of the large 'core' applications and middleware products that have been Web Services enabled.

- CSS (Customer Service Systems)

  CSS is a COBOL/CICS/IDMS-based application running on 29 mainframe MVS images. Serving 40 000 BT users, it holds data on 23 million customers and bills for 60+% of BT's revenue. CSS has 5000 distinct screens (maps) and 2000+ program modules and executes 240 million CICS transactions a day. This application supports over a thousand system interfaces including 600 services offering a 'tag length value' interface to scripts that run multiple COBOL transactions in sequence giving a course-grained business service interface.

- COSMOSS (Customer-Oriented Service Management of Special Services)

  COSMOSS is a single image CICS/COBOL/DB2 system handling the majority of order entry and order handling tasks for private circuits. Running 3m transactions a day, it is a key resource with a user population of 6000.

- CAMSS (Computer Aided Maintenance of Special Services)

  CAMSS is a single image IDMS/DC COBOL/ADSO system. It is the primary trouble ticketing system for private circuits. It can handle up to 4000 faults per hour and is available 24/7/365 with a hot stand-by fall-back machine. An outage of 1 hr can cost BT up to £1m in penalties paid to customers for non-resolution of faults within the service-level agreement. CAMSS holds data on more than 2m private circuits together with 500k pieces of customer equipment. It currently offers on-line access to some 50 services.

- STAA (Single Transaction Automation Architecture)

  STAA is a 'mid-tier' deployment platform for value-add services sitting on top of BT's legacy systems. It is a J2EE platform designed for the implementation of complex business rules/processes that can be reused across the enterprise. It is currently a low-volume, high-value platform offering.

3.1 CSS

To WS-enable CSS we provided a middleware solution that interfaced to the existing middleware message-based interface (MMBI) [1] scripting layer. The existing scripts produced an internal data model aimed at producing and consuming a tag length value string of data, for example:

RUN-OBJ;6;WS007A;CORRELID;10;0123456789;
TRACE;3;OFF;LOCN;2;LS;CTL-END;1;*;ACCT-NR;
10;01234567890; TEL-NR;11;01732464444;
FOLLOW-UP-REQD;2;NO;SUSPEND-END-DATE;
6;260603;END;1;*;

There is nothing in this format that would prevent it being represented as XML and so this is done in a non-invasive way and the necessary SOAP headers/envelopes and encoding attributes can be added in or stripped off, producing the following XML string as the Web Services equivalent:

```
<soap:Body>
<n:suspendDirectDebit xmlns:n="http://
www.bt.comcss/mmbi/WS007A/">
<mmbi>
<correlId>0123456789</correlId>
<trace>OFF</trace>
<location>LS</location>
</mmbi>
<acctNr>01234567890</acctNr>
<telNr>01732464444</telNr>
<followUpReqd>No</followUpReqd>
<suspendEndDate>string</suspendEndDate>
</n:suspendDirectDebit>
</soap:Body>
```

By comparison this new facility is called MXBI (middleware XML-based interface). The CSS mainframe accepts requests over MQSeries or TCP/IP sockets but not HTTP/HTTPS. It is necessary therefore to implement a simple protocol converter in the logical mid-tier to enable clients that only talk HTTP to invoke the range of services. The majority of access is via the WS-proxy in Fig 2. Security is also applied at this level.

In this way we are able to provide a Web Services interface to 600 scripts covering a large subset of CSS functionality without re-coding or changing the current services. The most obvious limitation of this approach is that the data returned is 'flat', i.e. it does not have logical groupings of related data elements in nested structures. In practice, therefore, we commonly take some time to re-engineer the scripts so that data can be represented more naturally, and to a more standard data model, than the existing scripts allow. This is a quick, low-risk activity because the business logic remains unchanged in the COBOL transactions.

3.2 COSMOSS

COSMOSS is a large CICS/COBOL system with a DB2 database and was engineered around the OO principle of encapsulation. COSMOSS has over 2000 database

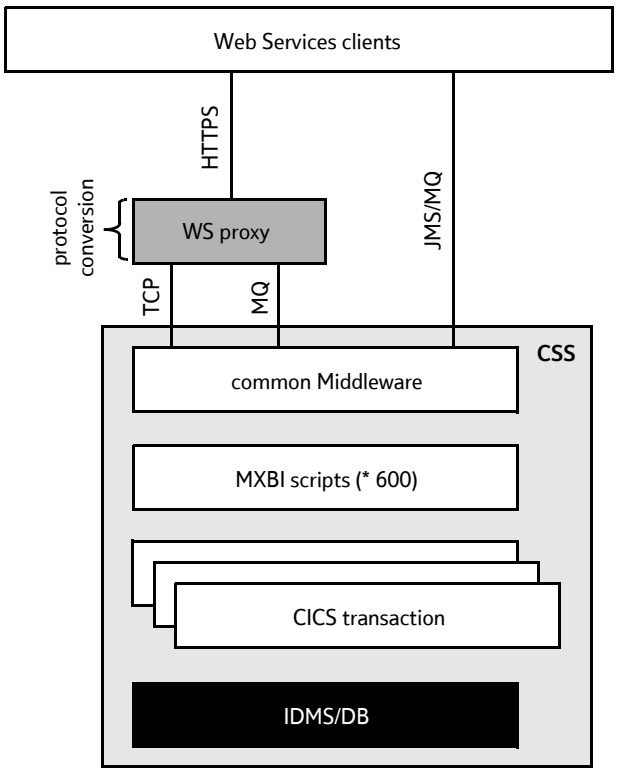

Fig 2 CSS access.

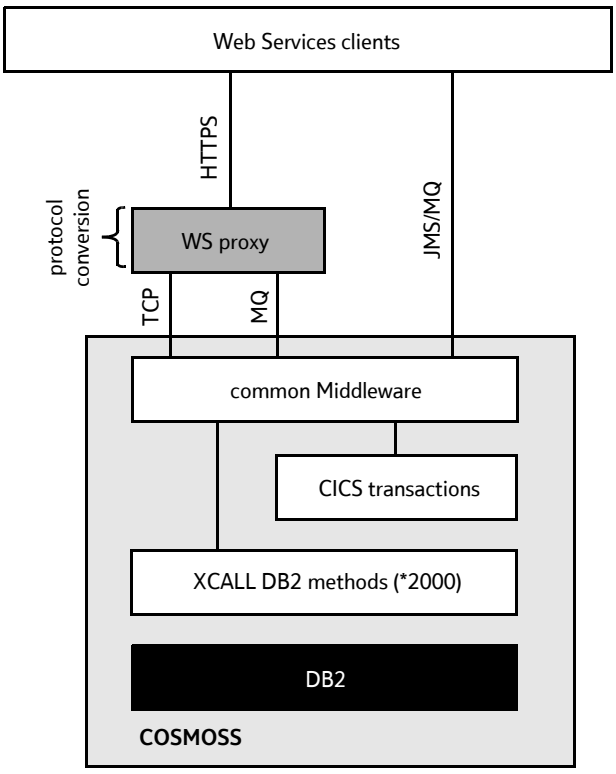

Fig 3 COSMOSS XCALL access.

methods implemented as COBOL programs that the business transactions call. Some of these methods are large, course-grained services implementing complex rules and reusable business logic. This encapsulation software is called XCALL.

To drive COSMOSS as a Web Service we have the same MXBI capability as for CSS, but for historical reasons there is nowhere near the number of existing scripts on which to base a WS capability. In addition to MXBI, therefore, we expose the database methods themselves as Web Services. Like the MXBI scripts, this can be done at a purely middleware level because we have a formal specification of what the interface is (via the schema and data dictionary). We can thus convert the data into XML according to a WSDL file that is itself automatically generated from the database method.

The COSMOSS processing is depicted in Fig 3.

Although there are 2000 potential Web Services on COSMOSS, we have converted less than 200 of these to a Web Services format and only anticipate circa 100 being used by more than one project. That said, the business case for implementing 'only' 200 interfaces, for little development/test/deployment cost in a fashion that makes it attractive and easy to use by a wide range of clients, is very compelling.

Challenges for Web Services-enabling of XCALL/COSMOSS have included making the Web Services interface 'pretty' and natural looking, given that the underlying method was designed to return rows/columns of segmented data. The techniques used to address this have included translating field names to meaningful data tags and restructuring repeated data into SOAP/XML arrays.

The Web Services proxy is again acting as a protocol converter with the majority of the Web Services-specific processing being performed on CICS by the COSMOSS middleware.

3.3 CAMSS

The existing on-line access to CAMSS was implemented over TCP/IP sockets interface with data being returned in the internal EBCDIC fixed-format copybooks that CAMSS uses internally. This is an unattractive interface for client systems to use and as a result CAMSS was normally hidden behind a mid-tier component that exposed a (slightly) more friendly access using DCE, CORBA or RMI. The problem with this centres around the fact that not everyone can call these APIs, as well as the extra cost of writing and deploying a mid-tier process to perform the translation.

Unlike CSS and COSMOSS, it was decided that we would not be able to make major changes to the CAMSS platform. Instead we invested time in building an infrastructure component that would automatically generate the WSDL from the internal COBOL copybook definitions. This was used to generate the translation

code on the mid-tier, that manipulates the fixed format data, into the correct SOAP XML structure to match the WSDL description (see Fig 4).

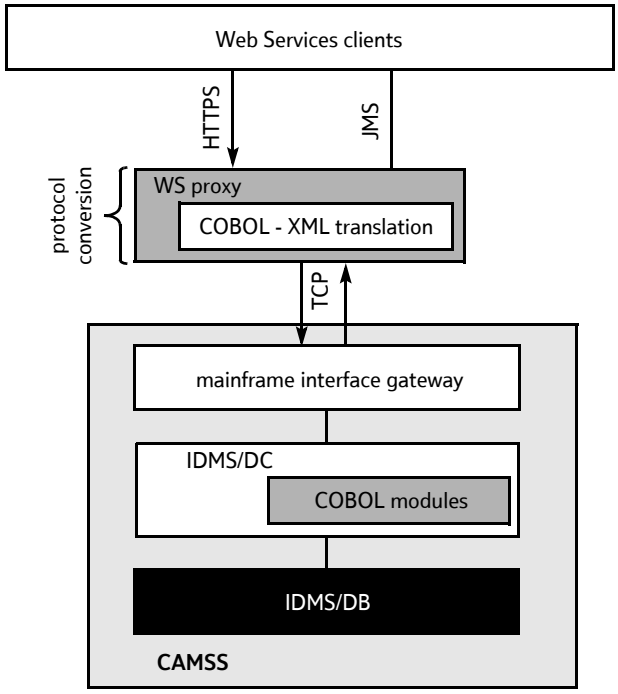

Fig 4 CAMSS WS access.

In this way the current set of existing transactions have been exposed as Web Services up front and, although a functional mid-tier component is required, it is not a hand-coded delivery and the service itself does not need to be re-tested since it has not changed.

3.4 The Web Services landscape

We have also worked on providing Web Services interfaces to DCE and CORBA platforms in BT utilising the same J2EE WS capability as is used for STAA above. There is also the capability (not yet in production) to drive our classic EAI hubs as Web Services in their own right. Together with an emergent use of Microsoft .Net as an integration platform, a considerable portion of the BT applications architecture is capable of both consuming and offering Web Services interfaces.

Figure 5 shows a range of the different platforms and technologies we have enabled with Web Services.

The key elements of our work in this regard have been to address the integration problem at a middleware level rather than at the application level. We try to hide the complexity of XML from the developers wherever possible and even in XML-aware applications we employ the concept of the 'SOAP bubble', a thin transparent layer of processing that performs all SOAP-specific processing required by a message.

The landscape (Fig 5) shows a common interface style to both legacy systems and mid-tier processes. This homogeneous style of interfacing allows us to best use each layer as appropriate. EAI hubs do not need to be used for every single exchange between two platforms, if it does not add value to the architecture. Neither do simple read-only or straightforward update processes need to be mediated by traditional mid-tier

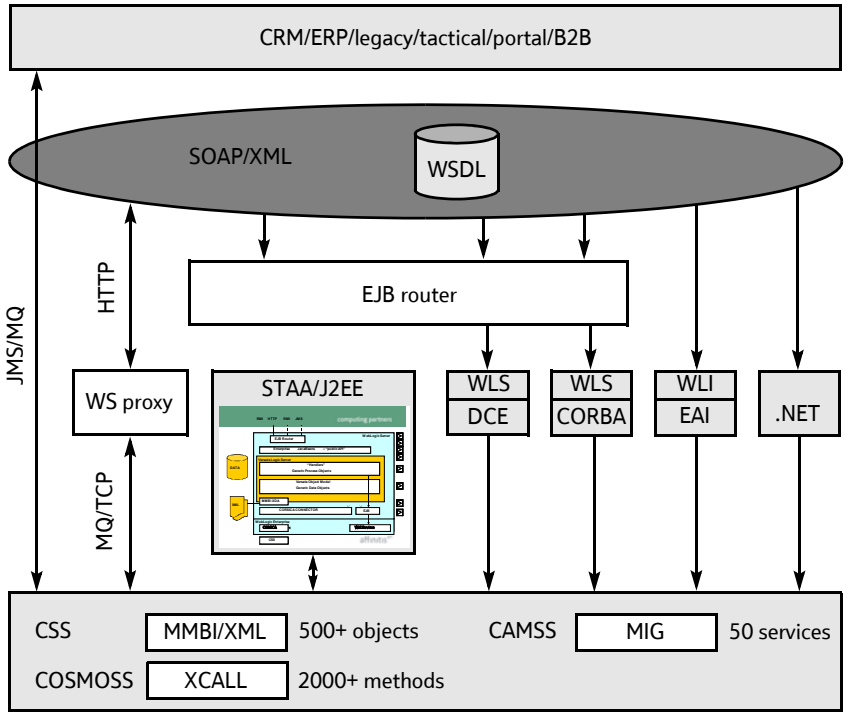

Fig 5 The Web Services landscape (unless explicitly detailed otherwise all data exchanges are WS enabled, SOAP over HTTP, and described by WSDL).

integration developments if no logic or business rules need to be implemented.

4. Interoperability

We selected Web Services for the heterogeneous interworking we required to address the range of BT's current applications estate. At an early stage we were made aware of the shortcomings of both the SOAP standard and the diverging interpretations of it by different vendors. Initially this was not widely recognised in the industry as a problem; the technology was immature and most people were only playing with Web Services, quite possibly in a totally homogeneous environment. The heterogeneous nature of our architecture meant that early on we recognised interoperability was going to be the most important issue for us to resolve, outweighing the more publicised concerns about security and performance.

To date we have implemented or evaluated over 20 distinct Web Services technologies and platform solutions — Apache SOAP, Axis, BEA WLS 6.1, WLS 7, WLP8, Cape Connect, Cape Studio v3/v4, MS .Net (c#, vb, C++), MS SOAP, Silverstream, SOAP:Lite, WASP (Systinet), WASP C++, GLUE (The Mind Electric), Sun JWSDP, gSOAP, PocketSOAP, Leif (Rogue Wave), Websphere (IBM).

This is only a small subset of the WS tool-kits available; for a more comprehensive list see the XMethods Web site [2].

As we have progressed we have tested these tool-kits against each other and against the services that we had already exposed. The result of this testing is that we have identified many interoperability issues associated with the SOAP implementations where each vendor has perhaps only implemented a subset of the standard. In an environment where their technology is used exclusively you will see no problems but trying to call such a service from a different technology may present difficulties.

To address this situation we have done two things. Firstly we have captured and publicised internally a list of 'Top Tips' for interoperability and used this to ensure that the Web Services we build or expose conform to a 'lowest common denominator' standard to ensure interoperability. Even conforming to this standard, it is considered necessary to validate a service against a representative range of tool-kits as well as validating any new technology or tool-kit that is brought in against the existing body of Web Services within the company. Secondly, therefore, we have evolved a facility called the 'Interoperability TestBench' which will perform both of these functions (Fig 6). Typically the process can be described as follows:

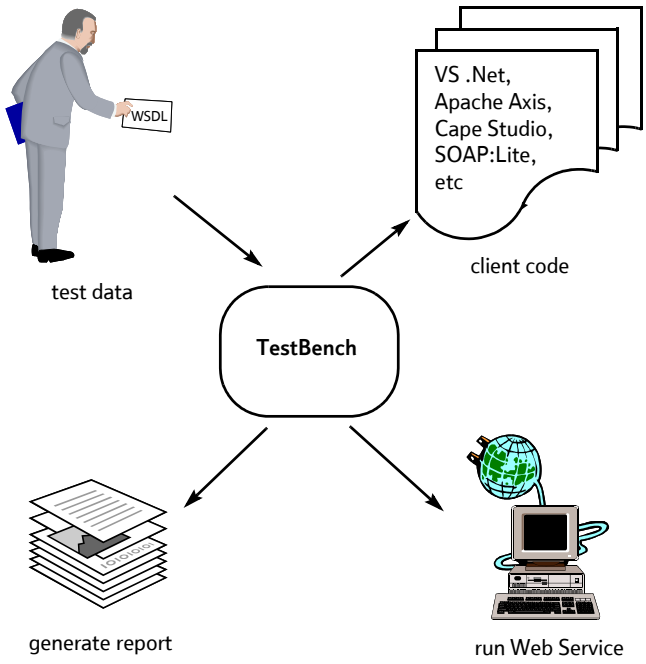

Fig 6 TestBench usecase.

- the customer presents the TestBench with a valid WSDL file describing their Web Service together with some test data that constitutes valid input/output values of the Web Service,
- the user selects a shopping basket of technologies against which they would like to test their service,
- the WSDL is imported into the TestBench and a data model constructed, mapping on to the input/output data structures as described by the WSDL,
- the TestBench then runs the WSDL through the range of tool-kits/technologies selected and produces the native stub code ready for use by client systems (any tool-kits that fail to process the WSDL file are flagged at this stage and are not processed any more),
- the TestBench then builds a client in the native language of the tool-kits that have been selected — this stage could very easily see clients being generated in Java, C#, C++ and Perl, etc (any client code generation failures are captured at this stage),
- the TestBench then feeds the client code with the test data and runs the Web Service for every tool-kit/technology that has successfully reached this stage (any data returned is stored),
- the test data is then finally compared with the real data returned and any discrepancies are highlighted,

- a report is produced detailing, for each tool-kit, whether the client could be generated, built and executed against the submitted WSDL — elementary performance statistics are also captured.

The other main use of the TestBench is to certify a particular technology against a set of Web Services. In this scenario the same tool-kit is used every time to generate, build and execute client code against a selection of Web Services that we know to exhibit interoperability issues with other technologies. These problem Web Services are called 'feature tests' and we continue to accumulate these feature tests as new services and tool-kits are put through the TestBench.

To the casual reader this may seem like a lot of effort, but we contend that this level of testing is necessary until the industry as a whole rigorously adopts all aspects of the SOAP standard or alternatively unites behind the Web Services Interoperability Forum [3]. To be WS-I compliant vendors must only implement those aspects of the SOAP standard that conform to the WS-I 'Basic Profile'.

BT has found the TestBench to be key to validating its own SOAP stacks/proxies, as well as aiding understanding of the reality behind Web Services standards. Within the enterprise we can restrict the client and server technologies that people are able to use. We will never be able to control all developments inside the enterprise though and we will have even less control over what our partners use to build clients to our external Web Services. Without such testing there is a risk of developing a service that some clients will not be able to use.

5. Security

Security is one of the last problem areas to be addressed by the Web Services community, and it is frustrating that such an important aspect has been so slow to develop. The standard in this area is now recognised as being WS-Sec [4] which offers the fine-grained message level security that distributed computing requires. For a detailed discussion of the relevance of this standard, see Kearney et al [5].

Unfortunately for BT, adoption of this standard is not widespread and is expected to be slow. Web Services and industry analysts, GIGA, claimed in March 2002 that it would be 18—30 months before adoption of the standard was widespread and mature enough to make its use feasible because of interoperability issues between vendors' implementations.

To date we have relied exclusively upon transport level security and the underlying security mechanism of the platforms on which the Web Services run. This type of technique includes the following.

- IP filtering

  This low-level technique involves the server only accepting requests from IP addresses that it knows represent valid client systems. As a security mechanism it is barely adequate even within a secure internal network because of the ease in which a rogue client could spoof the incoming IP address.

- Basic authentication over HTTP and HTTPS

  This technique involves a user ID and password combination passing in the clear or encrypted (HTTPS) over a TCP/IP connection. If the user ID and password are not validated, access to the Web Service is denied. Shortcomings of this approach is that the password is expected to be static. This means that if a dynamic password manager is employed to accommodate security requirements an alternative out-of-band mechanism is needed for resetting the password.

- Client certificates

  This is most likely to be used for machine-to-machine interfacing and involves the distribution of a managed time-allocated certificate that allows the client to authenticate itself to the server.

- Application level security

  This is typically used where there is an existing application being accessed that has its own security function. This technique normally sees the client send down security credentials in an initial request and be passed back a security token that must then be presented with each subsequent request. The shortcoming of this is that the security is now tightly coupled with the service definition itself and cannot be removed without changes to both client and server application interfaces.

The challenge for us, and other early adopters of Web Services, will be the migration to the WS-Sec standard and managing the support of multiple security solutions to the same Web Service.

6. Performance

The performance of Web Services is another area that has been of concern to potential users of the technology. This is understandable for a number of reasons:

- Web Services are associated with HTTP which is not considered a high performance protocol,

- the XML documents are typically 50—200% bigger than other wire protocols,
- the security solution in place may well necessitate the use of HTTPS which can be slow given the large packets of data needing to be encrypted,
- Web Services are frequently implemented using languages such as Java/Javascript/Perl, etc, which are slower than other fully compiled languages,
- early Web Services, hand coded by users, exploited inappropriate techniques such as XML schema validation and XSLT processing which are very slow operations.

Indeed it is possible to find suggestions that Web Services are 20—40 times slower than non-Web Services solutions [6].

We have invested some effort in identifying the overhead of using Web Services and have found a very different experience for the type of Web Service that we are using inside the company.

Space does not permit detailed performance statistics to be presented here but for a typical on-line service taking 2 sec to run and returning 5 kb of data, an overhead of 25% may be attributed to the use of WS technology.

Even for the simplest of services one can expect the use of Web Services tool-kit technology at either end to add on up to 200 ms to the overall end-to-end response time. Genuine real-time operations will find this an unacceptable overhead and we do not envisage Web Services being used for general-purpose network control.

There are many factors that will effect performance but the two greatest impacts we have observed is the choice of tool-kit and the implementation of security.

Individual Web Services tool-kits and technologies have been observed to be extremely slow in certain circumstances (e.g. from the selection of Java tool-kits we have tested, the slowest takes 3 times longer to do the same elementary service as the fastest). This type of result, though, should be put down to the immaturity of the offerings rather than generally dire performance for Web Services.

The implementation of security is also responsible for some degraded performance, but generally performance is acceptable for most user-driven on-line implementations. As the use of HTTPS for large messages will always add a significant delay, it is important to appreciate this overhead through testing out the technologies that you plan to use.

In summary then, we believe that Web Services performance is 'good enough' for on-line synchronous systems integration. It is not currently an appropriate solution for genuine real-time integration requirements or for extremely high volume publish-and-subscribe implementations. The benefits achieved through the use of Web Services technology can outweigh the performance degradation, and, in existing scenarios involving the use of Web browsers and HTML interfaces, the delays incurred could even be considered minimal.

As stated earlier, some of the real-life uses of Web Services inside BT have allowed us to remove layers of code and/or middleware, thus actually improving the performance of the interface (see section 8.1). Our experience in doing this leads us to make the following recommendations:

- avoid XSLT processing if you are working at the XML level,
- similarly, be cautious about performing schema validation if this is not required,
- the use of security options can have a major impact on performance,
- ensure the tool-kit technology is working over HTTP1.1 not HTTP1.0,
- benchmark your selection of Web Services technologies against each other to highlight inefficiencies,
- exploit asynchronous patterns where appropriate.

7. UDDI and service discovery

UDDI is, in effect, a registry or directory for publishing and finding Web Services [7]. Unlike some directories UDDI does not actually describe or point to individual Web Services. This is the role of the WSDL. The UDDI points to the WSDL when the correct service has been found by navigating the directory structure to the correct category. As such it is not an essential part of the Web Services in the way that SOAP, XML and WSDL are.

Universal description, discovery and integration is the least implemented of the 'core' Web Services technologies. The Cinderella nature of this technology to date can be attributed to the fact that few companies are advertising services outside the enterprise for general consumption and therefore the need for discovery is much reduced. Likewise inside the organisation there is not the likelihood of a multiplicity of identical services that a UDDI implementation could

help to describe. If an enterprise does not have a significant body of Web Services to publish and categorise, the requirement for UDDI is not obvious.

Pragmatically it was accepted that we may not actually need an internal UDDI instance until we had achieved a 'critical mass' of services across a range of business areas. Strategically though it was recognised that to delay implementing a UDDI service for our internal Web Services would mean a risk of being unable to support and administer the service immediately when it was required.

To this end we elected to implement a UDDI registry that describes our internal Web Services before the number and complexity of these services grew to the point where a registry component was required.

A detailed technical evaluation of UDDI offerings resulted in BT selecting the Microsoft UDDI as the preferred solution. A key factor in this decision was the ability to create custom categories in the UDDI that we could use to describe the business areas and functions of the Web Services we were creating. This remains a weakness in many UDDI offerings currently available.

Our experience has shown that the large organisation may need more than one categorisation scheme or means of finding the service(s) being published. It may be beneficial for services to be categorised in different ways depending on the knowledge and viewpoint of the designer/developer using the service. For example, services may be categorised around business areas, functional areas or product ranges. This flexibility cannot be achieved if the UDDI offering only allows services to be published according to predefined categorisation schemes.

There is also the question of how such services may be published outside the enterprise when needed for integration with customers and trading partners. BT is also currently involved in a work package to implement the eTOM (electronic telecommunications operations map) on UDDI, although this does not achieve the fine level of granularity to describe services in such a manner that developers/solutions designers are able to quickly locate the right service.

In order to allow Web Services to be discovered at any stage of their life cycle, we have chosen to implement development, test and production UDDI instances. In this way the designer only wanting to discover services that are already deployed may be able to do so.

8. Case studies

8.1 BT.com

BT.com is BT's primary customer portal and offers many customer services such as 'View my bill', 'Friends and Family', etc. By using these services on line the customer is saving BT money because of the high cost of operator-assisted services. It is estimated that to change a customer's 'Friends and Family' numbers via BT.com costs 3p compared to £3 for an operator-assisted call. It is important therefore that customers are encouraged to use this service and that they receive a good experience when they first use BT.com.

In order to use BT.com for these types of service it is necessary to register as a BT.com user. The process involved an off-line validation of the customer followed by an e-mail being sent to the customer with their BT.com pass code, which they then used to access the services they required. Statistics showed, however, that people were not returning to the site after the pass code had been sent out. This meant that BT was losing the potential cost savings of on-line transactions, not building the on-line community as fast as it could and not offering the customer the best user experience because they were not making a return visit.

The requirement then was to allow the customer immediate access to the BT.com services such that we retained their custom, benefited from the cost savings and offered the best experience possible on their first visit to BT.com.

8.1.1 Proposition

BT Exact proposed an on-line validation of the customer, offering real-time confirmation of their identity to enable the on-line dialogue with the customer to continue and let them use the BT.com services straight away.

8.1.2 Solution

It was agreed that the best approach was to use the customer's billing information as proof of identity. The printed bill contains a 2-character check-digit field that is only present on the printed bill itself. If this information could be retrieved on-line and compared with the data supplied by the customer, then we had enough proof of identity to allow the customer to continue their dialogue. We were able to provide this information with a Web Services interface to CSS to retrieve the billing data while the customer was on-line.

8.1.3 Results and customer benefits

The solution was implemented in a very rapid time-scale and for low cost to the BT customer. The successful implementation of this has meant that a far higher take-

up of BT.com services has been realised. Real-time validation for 95% of calls is taking less than 1 sec.

By selecting a lightweight Web Service for this requirement, BT.com was able to take delivery of the service within 1 week and able to go live within 5 weeks of the initial request being made. Alternative costings were made and it is estimated that this was implemented in less than half the time of a conventional mid-tier deployment and for 25% of the development costs of hand coding a mid-tier solution.

8.1.4 Technology involved

The main components of the solution are shown in Fig 7.

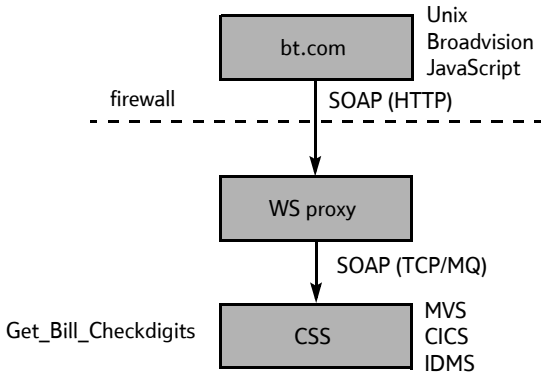

Fig 7 On-line trust.

8.2 SCORe

Project SCORe (Service Consolidation and Operational Revitalisation) is a BT Retail initiative aimed at reducing costs and increasing customer satisfaction with the way their contact with call-centres is handled. One of the problems identified was the system's complexity in retrieving all data relative to a customer's contact because it is held in multiple back-end databases. This requires separate calls to these systems using different access paths and technologies.

Part of the solution for BT Retail was to channel all of this contact through their strategic CRM (customer relationship management) product. The CRM platform, however, was not an appropriate place to implement complex business rules or cross-platform integration solutions.

Phase 1 of SCORe required access to CSS, COSMOSS and CAMSS. These are three applications on very different database and TP Monitor combinations — integrating with all of them would historically have meant using three different protocols and data models.

It was suggested to BT Retail that life-cycle savings and a more elegant architecture could be achieved if a single interface style was used across all three systems. CSS and COSMOSS already had a WS interface and so the CAMSS interface (described above in section 3.3) was developed for this deployment. The initial deployment makes use of two services on CAMSS, two services on COSMOSS and six services on CSS.

The whole piece is offered as a coherent set of services (Fig 8) through the STAA platform (described above in section 3), thus providing the capability to integrate multiple heterogeneous platforms via Web Services.

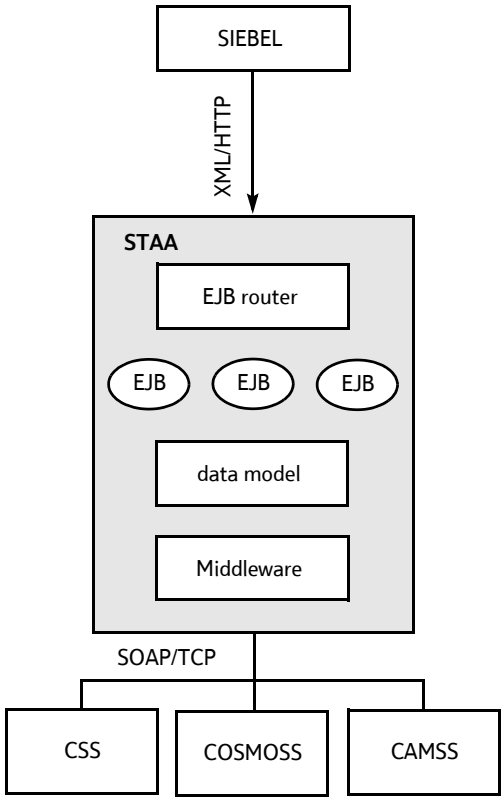

Fig 8 SCORe architecture.

The main benefit of this deployment is that the integration layer of the STAA platform that implements the common data model is able to access services on CICS, DB2 and IDMS/DC in a single consistent fashion. This means that integration activities can be developed more quickly, and ongoing support is reduced with fewer integration technologies to be maintained.

8.3 Service provider gateway

Service provider gateway (SPG) is a BT Wholesale application that allows restricted access to certain functionality in the BT OSS stack to external service providers. Typically this is around trouble ticketing and line testing operations. Many of the functions required by SPG on the back-end systems were accessible via a DCE mid-tier. DCE is effectively an obsolete technology and BT is actively removing it from the infrastructure,

having been served notice by our DCE provider that support will be withdrawn.

When SPG required new features for their access to CSS it was decided that we would offer them a Web Services interface to the functions they require. Most of the services were read-only or simple update transactions with little business logic needed. These services would be implemented as Web Services direct to the CSS application, whereas any service that required complex processing or application of business rules would be implemented on the STAA platform as a J2EE-based Web Service.

SPG is a Unix-based application with an Oracle database and the majority of the business logic written as PL/SQL stored procedures (see Fig 9). The software levels that SPG were at and the environment that they used prohibited us from exploiting a Web Services tool-kit to effect the client interface, but the developers were able to call the Web Services through manipulating XML structures directly and send these out as requests over HTTP. This demonstrates the flexibility and the near universal access that the Web Services model exposes. The benefits of this approach are:

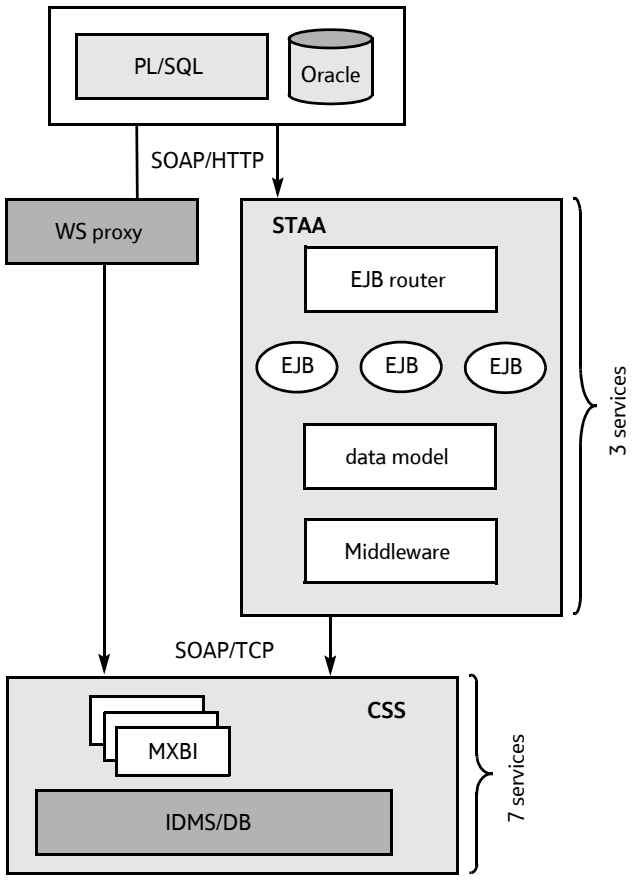

Fig 9 SPG architecture.

- access to non-native services being made available as Web Services means that heterogeneous platforms are able to invoke them,
- the services could be individually deployed on the most suitable platform and still be accessed in a consistent manner by the client,
- being able to deploy Web Services directly on the mainframe rather than coding a needless mid-tier solution saved the customer circa 80% of their standard development costs alone,
- an outdated technology (DCE) could be removed from the architecture,
- with the majority of services implemented directly on the back-end system, performance of the application is greatly improved.

9. Conclusions

BT Exact is now some way into a programme of promoting Web Services as the primary technology solution for internal applications integration. This approach has been proved across a wide range of platforms and technologies and has delivered real business benefit in delivering a better, cheaper, faster integration capability. Web Services is a credible model for large- scale systems integration and we are confident that it will serve BT well and give our legacy systems an extended lifetime of participation in our distributed systems architecture.

There is wide industry support for Web Services. However, it is still an evolving set of standards and Web Services adopters must be cognisant of a number of issues while navigating their way through the technologies and planning their strategy. Inter-operability, in particular, must not be assumed and there is a need to prove the interoperability of Web Services until initiatives such as the WS-I begin to deliver conformance to the SOAP standard.

The core set of Web Services technologies are now stable and this will mean greater stability and less diversity in the different implementations of these standards that are available. There are new standards emerging all the time that will extend and enhance the Web Services capability, notably Web Services reliable messaging and choreography/orchestration. The adoption of these standards by the market-place will further promote the use of Web Services as an industry-strength solution for comprehensive enterprise integration.

Acknowledgements

The developments and strategies described in this paper are the result of the dedication and expertise of

the BT Exact Web Services Integration Team, in particular, Newland Andrews, Ken Bugbee, Paul Downey, Mike Gibson, Pete Harris, Chris Hipson, Colin Jones, Peter Kennedy, Dave Prout, Adrian Smith and Terry Wyatt.

References

1 Calladine J: 'BT Middleware — software as infrastructure', BT Technol J, 15, No 1, pp 135—146 (January 1997).

2 XMethods — http://www.xmethods.com/

3 Web Services Integration — http://www.ws-i.org/

4 OASIS — http://www.oasis-open.org/committees/

5 Kearney P et al: 'An overview of Web Services security', BT Technol J, 22, No 1, pp 27—42 (January 2004).

6 Mclaren R: 'Real World Web Services Deployment', presentation at Web Services Architecture 2002, reproduced in Web Services Strategies, 1, No 4 (December 2002).

7 UDDI — http://www.uddi.org/

Jon Calladine is the manager of the Web Services Integration team in BT Exact. He holds a BA in Philosophy and Artificial Intelligence from Sussex University, and joined BT in 1987.

He initially worked as a systems programmer on some of the core systems comprising today's more 'venerable' legacy applications.

In 1991 he moved into systems integration and has subsequently worked on many different integration technologies for both in-house and third-party products. He championed the introduction and roll-out of MQSeries as BT's core asynchronous message queuing technology and was chairman of the UK user group for 3 years.

Since 2001 he has promoted the use of Web Services technologies within the integration infrastructure as a strategic direction.

He currently represents BT in the WS-I (Web Services Interoperability) forum.