An evaluation of four patterns of interaction for integrating disparate EBSs effectively and easily.

Authors

Abstract

Organisations use Enterprise Service Busses (ESBs) to support enterprise application integration. For a variety of reasons – mergers and acquisitions, geographically distributed enterprise units, distributed governance, scalability – enterprises sometimes need to acquire multiple, disparate ESBs and enable the applications that each one supports to interoperate. However, currently, no standard architecture exists for integrating multiple, disparate ESBs. To begin to address this problem, four candidate disparate ESB integration patterns – directly connected, web services, homogeneous messaging middleware, and message bridge – were identified from the enterprise application integration literature and tested for their effectiveness in integrating multiple, disparate ESBs. Each pattern was applied in two different scenarios: loan broker request, and inter-divisional messaging. In each scenario a number of enterprise applications were integrated using three disparate ESBs: Oracle Service Bus, Apache ServiceMix, and Mule ESB. The experiments were designed to test how well the different patterns supported effective integration of different ESBs. The results indicate that the web services and homogeneous messaging middleware patterns are the best for integrating disparate EBS effectively and with minimal difficulty. In addition, it was discovered that the degree to which ESB integration could be achieved depended upon the number of ESBs being integrated, the relevant skills of the integration team, and the types of the ESBs. The results may be of practical benefit to the communities engaged in enterprise application integration research and practice.

Keywords: ESB, disparate ESB, ESB integration pattern, web services, messaging middleware

1. Introduction

ESBs are used to support the integration of disparate enterprise applications within a business organisation (Moradi and Bahreininejad, 2010): for example, legacy applications, commercial off-the-shelf (COTS) applications, and multi-tier applications developed in a variety of programming languages, running on a variety of operating systems, within a variety of machines. Typically, an ESB will enable event, data, and command messages (Hohpe and Wolfe, 2004) to be routed asynchronously between such applications to support application interoperation. But, in addition to message routing, ESBs also provides other services: for example, message transformation and enhancement, security, monitoring and management, and business process control (Rademaker and Dirksen, 2009). And use of, and interest in, ESBs within enterprises worldwide is high as evidenced by a recent Gartner survey (Vollmer, 2011).

Business organisations may be structured in a variety of ways: single global company, geographically distributed, multiple business divisions, store/branch networks and hierarchically (Nott and Stockton, 2006). For some of these structures, for example for an organisation comprising multiple business divisions, distributed governance is often employed. Here, governance "refers to the set of services policies and best practices that allow IT organisations to effectively control the definition, creation, use amendment, and retirement – the life cycle – of business processes and the constituent parts from which they are composed." (Nott and Stockton, 2006). In addition, there are other cause of distributed governance within an organisation including growth through acquisition and mergers, the use of a franchise or cooperative business model, and government regulatory requirements (Keen, Bond, Denman and Husek, 2005). And a different ESB may be used to support each autonomous governance unit within a business organisation in order to enable different policies, for example different security policies, to be enforced.

Now "some consumers in one domain may request services provided by another domain" (Powell, 2008). Schulte estimated in 2007 that "by 2012, more than 67% of all SOA composite applications and processes will involve one or more services outside of their domains" (Schulte, 2007). But to achieve this, applications hosted on each ESB need to interoperate easily.

However, it is not clear what is the best way to support such interoperation; in other words it is not clear how to integrate multiple, disparate ESBs belonging to an enterprise. Keen et al. (2005) affirm that "integrating two or more ESBs is subject to some of the same integration patterns as connecting two or more applications", and, further, that "it seems logical that integrating ESBs should follow best practices similar to those for EAI" i.e. for Enterprise Application Integration. Here, though, there are a number of possible integration patterns, and it is not clear which pattern or pattern would be most suitable for integrating multiple, disparate ESBs. Consequently the authors carried out a series of experiments with four integration patterns imported from the EAI domain – direct connection pattern, homogeneous messaging middleware pattern, message bridge pattern, and web services pattern – using two real world scenarios in order to test empirically the hypothesis that one or more of the four patterns, or a combination of the patterns, will enable multiple disparate EBSs to be integrated easily and effectively.

This paper, based on Nwakacha's MPhil thesis, which was supervised by Green (lead) and Beeson (second), summarises and extends the results of that work. It is structured as follows. Section two provides a description of the EAI and ESB concepts. Section three presents the four integration patterns. Section four describes the two real world scenarios: a loan broker scenario and an inter-divisional communication scenario. Section five lists the eight experiments, identifies, explains and justifies the choice of experimental variables, and describes how the experiments were run. In section six, the results from the experiments are analysed. And section seven concludes the paper with a summary, and a restatement of

the main findings; it also discusses the significance of the results and the limitations of the work.

2. Integrating enterprise applications

As Alonso et al. (2004) point out: "During the late 1980's and 1990's, enterprises increasingly relied on software applications to support many business functions, ranging from "simple" database applications to sophisticated call centre management software or customer relationship management (CRM) applications." These enterprise applications and others like them – for example, legacy systems like payroll systems, and ERP suites (Papazoglou and Ribbers, 2006) – were generally stand-alone systems, sometimes distributed, but frequently disparate with respect to implementation language, operating system, interfaces supported, security requirements, data formats and so on (Alonso, Casati, Kuno and Machiraju, 2004). They were 'silos, where "islands of automation" and mission critical data were "locked" within disparate and ad hoc legacy systems" (Papazoglou and Ribbers: p 498). But, to support business processes, companies needed to integrate their internal enterprise applications, and later they needed to integrate with customers and suppliers across international boundaries (Papazoglou and Ribbers, 2006, p. 499; Alonso, Casati, Kuno and Machiraju, 2004). In theory this could have been done manually with the output of one application being entered by hand into the next application. However, this would have been an error prone and inefficient process. What was needed was a way to combine all the steps comprising a business process "into a coherent and seamless process" that could be enacted automatically (Alonso, Casati, Kuno and Machiraju, 2004).

Enterprise application integration (EAI) is the name used for the technologies and techniques that comprise the first main solution to this integration challenge. Papazoglou and Ribbers have written extensively on EAI. In (2006, p. 502) they indicate that the goal of EAI is: "to eliminate islands of data and automation caused by disparate software development activities and to integrate custom and package applications"... "to drive operational efficiency within organisations".

Early EAI comprised point-to-point direct connections between enterprise applications. Messages sent by one application to another needed to be translated into the format "understood" by the other; and vice versa for replies. This meant that the number of possible connections between applications was potentially n(n-1) [where n was the number of applications]. This represented an exponential growth in the number of connections, and entailed an increasingly severe maintenance burden.

In the hub-and-spoke topology, a later EAI technology, "a central node manages all interactions between applications" (Papazoglou and Ribbers, 2006, p. 509). The hub sits centrally between all enterprise applications and performs all routing of messages, translations and transformations. The number of potential connections between all connected applications is now 2n [where n is the number of applications]. The impact of

change is much reduced from the point-to-point topology because all changes are now made at the hub. But there are problems with this topology: Chappell notes that “BPM tools that are built on top of a hub-and-spoke topology can’t build choreography or business processes that can span departments or business units” (Chappell, 2004, p. 33). And also that it “may be limited by the underlying MOM [Message Oriented Middleware] in its ability to cross network LAN [Local Area Network] segment boundaries and firewalls.’’

According to Chappell (2004, p. 49), “Hub-and-spoke EAI brokers could get as far as corporate boundaries, but were not really built for scaling beyond that”. However, many organisations had become “extended organisations” (Chappell, 2004, p.51): their business processes extended beyond their corporate boundary due to, for example, mergers, acquisitions and collaboration with supply-chain partners (vendors and customers). Pressure to support such extended enterprises led to the development of the Enterprise Service Bus (ESB) concept and its constituent technologies and the capabilities they enabled.

Chappell (2004; p. 35) states that the key features of ESB are that they are based on message oriented middleware (MOM); that they are also based upon standards; that they provide support for web services and service oriented architectures (SOA), and that they support asynchronous messaging. He writes that “these collectively form an architecture for a highly distributed, loosely coupled integration fabric to deliver all the key features of an integration broker, but without all the barriers” (Chappell, 2004, p. 35). The standards that Chappell refers to include, for example, Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL) for web services, Extensible Stylesheet Language Transformation (XSLT) for message transformation, Java Message Service (JMS) for reliable messaging and Java EE Connector Architecture (JCA) for enterprise application adapters. Papazoglou (2012, pp. 262- 264) presents a longer list of an ESB’s capabilities that includes: a communications infrastructure supporting various abstractions like publish and subscribe, dynamic connectivity, topic- and content-based message routing, transformation “between different data formats and messaging models”, service enablement via SOA, endpoint discovery with multiple quality of service (QoS) capabilities, support for service orchestration and long-running processes, transaction management, integration, management and monitoring, and scalability. An ESB essentially supports an event-driven SOA. Enterprise application events “trigger asynchronous messages that are sent between independent software components”, i.e. other enterprise applications, “... by abstracting away from underlying service connectivity and protocols” (Papazoglou, 2012).

Returning to extended enterprises - where companies have merged with other companies, or bought other companies, or collaborate with other companies in supply-chain networks, or have geographically distributed, autonomous business divisions or units - it will often be the case that a number of ESB products, each possibly of a different type, will be in use. Schulte (2007) predicted that 80% of large organisations would have ESBs from more than three vendors by 2009. However, in such cases it will still be necessary for enterprise

applications hosted in one part of the extended enterprise by one ESB to integrate with other enterprise applications, hosted by other, possibly disparate, ESBs in other parts of the extended enterprise. In such cases there is a need to link the set of EBS into a federated ESB topology. “A federated ESB solution is used to form a virtual network of trading partners across industries and services able to take advantage of the wider range of options and partnering models” (Papazoglou, 2012, p.268).

However, currently there is no standard way to link ESBs into a federated topology and the research literature on this topic is sparse. To begin to address this problem, the researchers conducted a series of experiments in order to assess the extent to which each of four integration patterns could be used to integrate easily and effectively three disparate ESBs deployed in two real-world scenarios. This outcome of this work is described in the sequel. This begins in the next section, which describes the four integration patterns.

3. Four integration patterns

Keen at al assert that: “Integrating two or more ESBs is subject to some of the same integration patterns as connecting two or more applications; integration déjà vu” (2005, p. 87). They reasonably suggest that one important implication of this assertion is that “it seems logical that integrating ESBs should follow best practices similar to these for enterprise application integration (EAI)”. They propose three patterns for integrating ESBs:

- Directly connected ESB pattern
- Brokered ESB pattern
- Federated ESB pattern

In the directly connected ESB pattern, two or more ESBs are directly linked to each other as figure 1 illustrates.

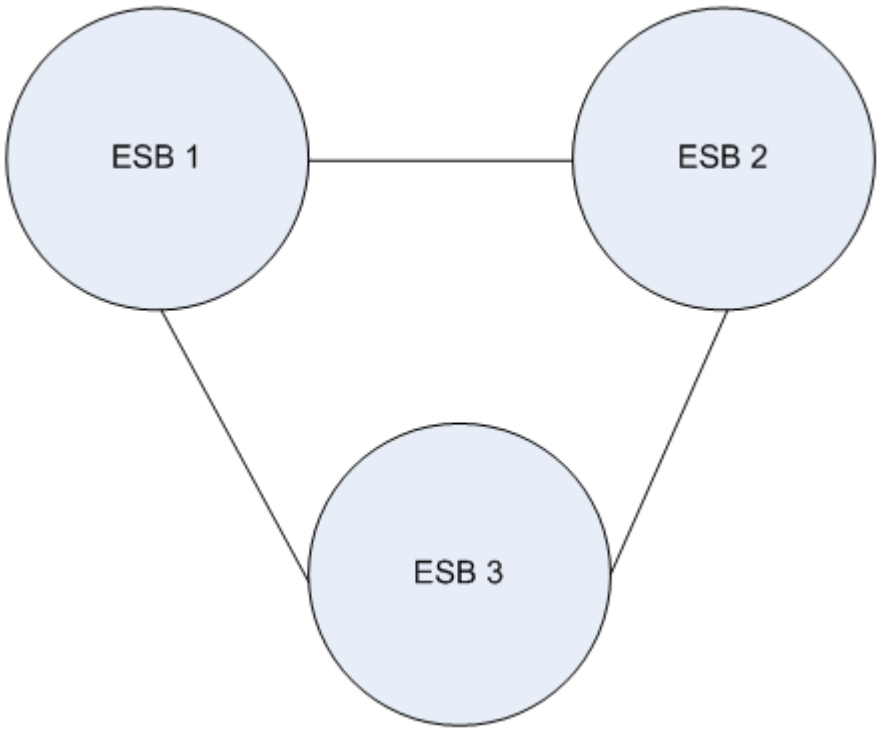

Fig. 1: directly connected ESBs

“Each ESB maintains its own namespace directory, administration and security services” (Keen, Bond, Denman and Husek, 2005, p. 96). Service providers hosted by one ESB are mapped directly to service consumers hosted by other ESBs, and the end points contain knowledge of each other. So, for example, an ESB hosting a service consumer “knows” which ESB is hosting a desired service provider, what protocol to use, and what message format to use in order to route a request (Keen, Bond, Denman and Husek, 2005, p. 96).

In the brokered ESB pattern, two or more ESBs are connected to a brokered ESB gateway rather than directly to each other as figure 2 shows.

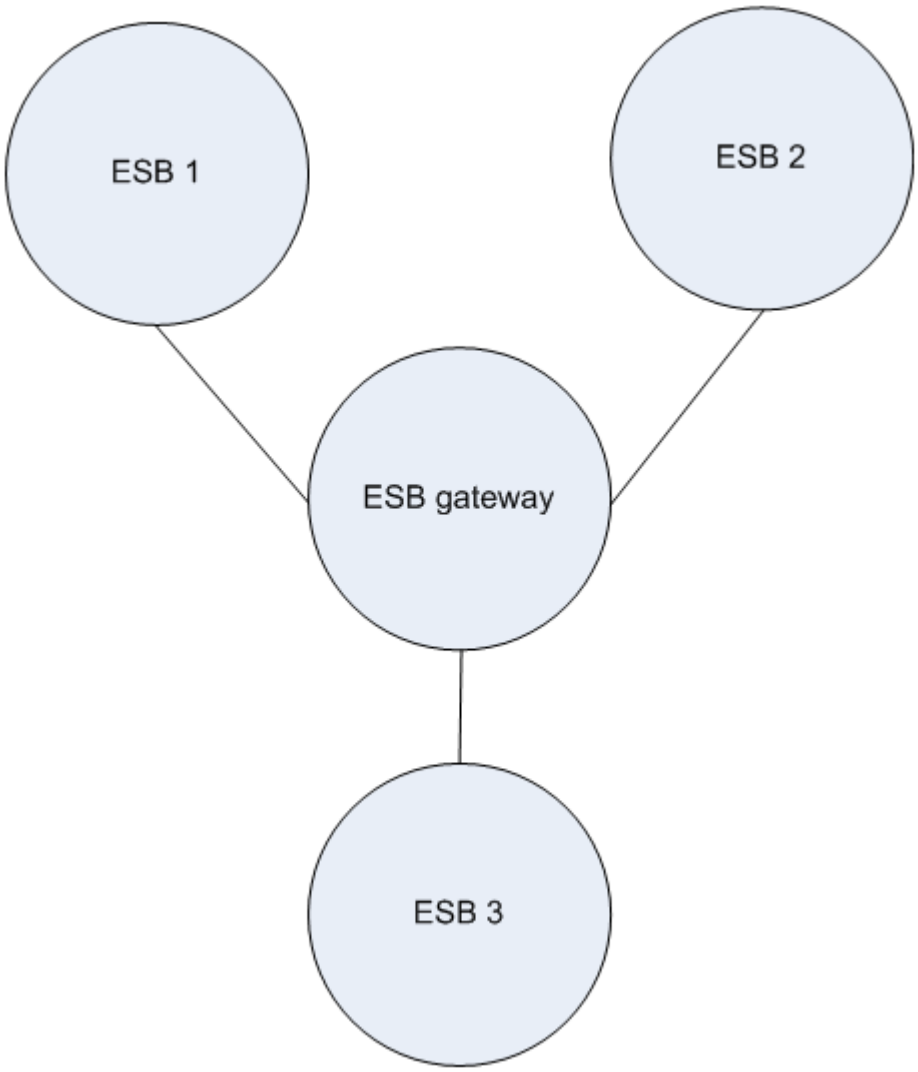

Fig. 2: brokered ESBs

The brokered ESB gateway hosts neither service providers nor service consumers; it just provides a routing service to integrate attached ESBs and their hosted enterprise applications. Keen at al. (2005) note that it “often connects heterogeneous ESBs”. Since all of the attached ESBs are linked just to the brokered ESB gateway rather than to each other, the number of links no longer grows exponentially as new ESBs are added: so the potential maintenance task is made considerably less difficult.

The federated ESB pattern is similar to the brokered ESB gateway pattern (see figure 3), but it also “adds orchestration of service consumer requests that span multiple ESBs, multiple service providers, or both” (Keen, Bond, Denman and Husek, 2005, p. 100).

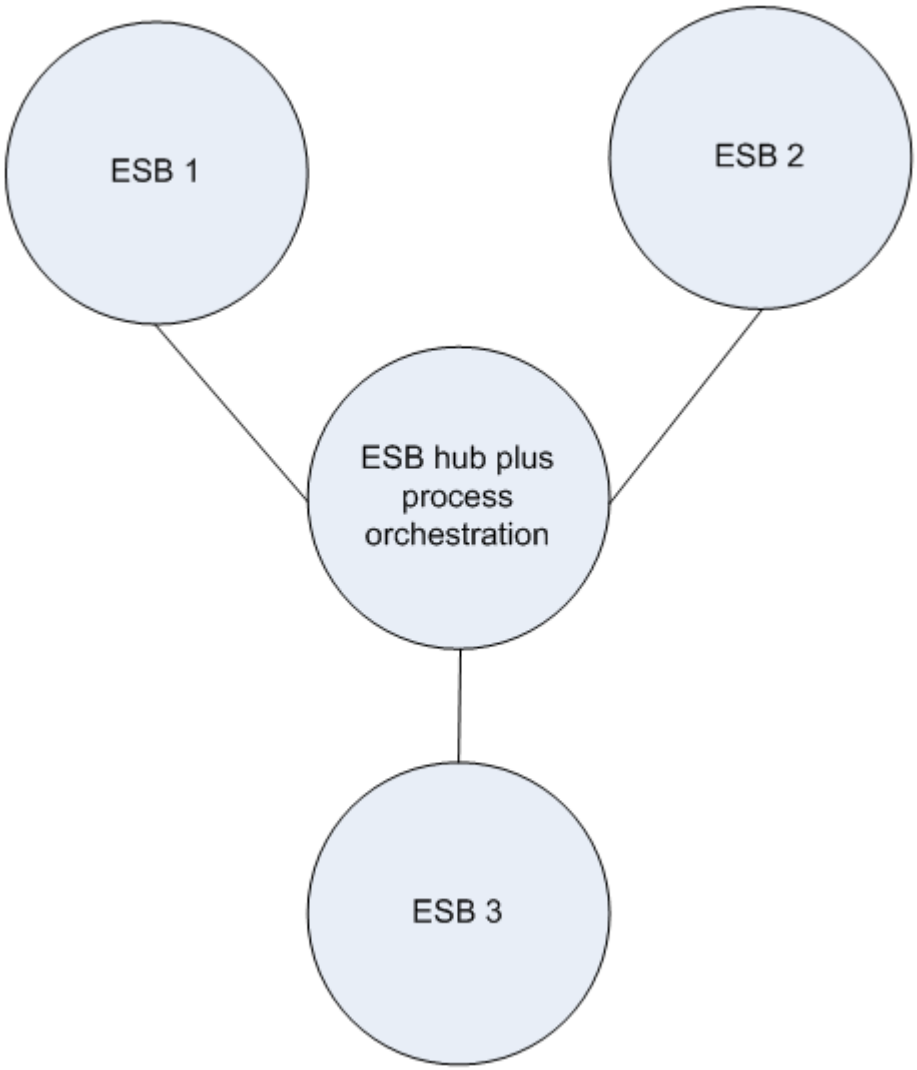

Fig. 3: federated ESBs

“A service consumer can make requests that require resources from multiple service providers on multiple ESBS”... and the federation component “will control the coordination of multiple service providers”.

For the empirical research described here, four integration patterns were used that were based upon Keen’s patterns (Nwakacha, 2011). (Keen’s source pattern is named in parentheses.)

- Direct connection pattern (directly connected ESB pattern)
- Message bridge pattern (brokered ESB pattern)
- Homogeneous messaging middleware (brokered ESB pattern)
- Web services (federated ESB pattern)

The direct connection pattern is based directly on Keen’s at al.’s directly connected ESB pattern with connection adapters being used to mediate between disparate ESBs. The message bridge pattern is based upon Keen’s brokered ESB pattern. Nwakacha quotes

Hohpe and Woolf (2004, p. 134): “when a message is delivered on a channel of interest in one messaging system, the bridge consumes the message and sends another with the same contents on the corresponding channel in the other messaging system.” Here, “messaging system” refers to a set of enterprise applications and their ESB host. The homogeneous messaging middleware pattern is also based upon Keen’s brokered ESB pattern, but a message queue platform is used instead of an application server (Nwakacha, 2011, p. 19). “The set up for this pattern includes a server and a client platform. The client platform is configured on all the ESBs and communicates with the server through the messaging protocol. In essence, the ESBS will act as the messaging clients to the messaging server. The messaging server hosts a queue/topic manager which is a container containing queues or topics. To have access to the queues/topics, client must go through the queue manager” (Nwakacha, 2011, p. 20). Figure 4 shows the connection established between a messaging client and a messaging server.

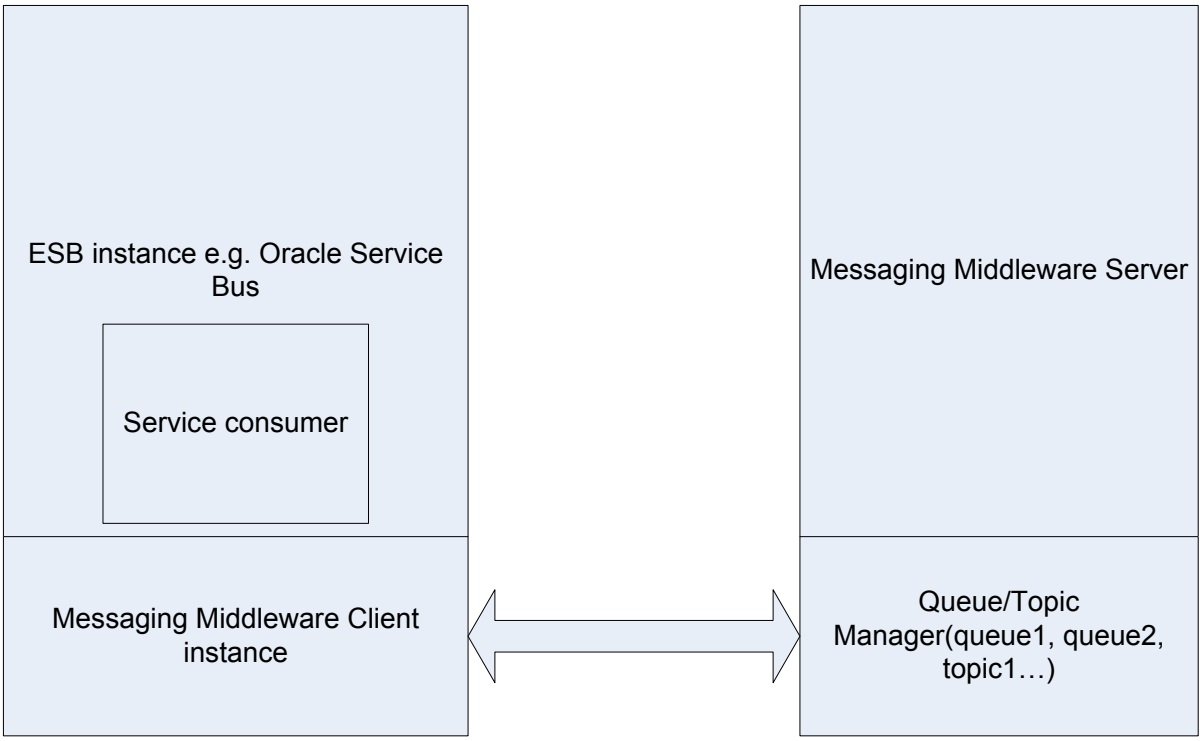

Fig. 4: messaging middleware communications

The web services pattern is based upon Keen et al.'s federated ESB pattern (2005). In this pattern, the service provider's interface is exposed as a web service, and the service requestor, possibly hosted on a different and disparate ESB, subscribes to it using, for example, SOAP over the Hypertext Transfer Protocol (HTTP). Such services can be orchestrated using, for example, a business process management system, in order to provide a composite solution.

4. Scenarios

Two business scenarios were used as case studies in the experiments that investigated the ease and effectiveness with which disparate ESBs could be integrated: a loan broker scenario that features a customer using a loan broker to obtain a bank loan, and an inter-divisional messaging scenario that simulates a company with multiple interacting divisions (Nwakacha, 2011, pp. 26 – 35). The loan broker scenario is described here; but the reader is referred to Nwakacha's thesis (2011) for details of the inter-divisional messaging scenario.

The loan broker scenario is based on the example provided by Hohpe and Woolf (2004, p. 363)

"Because contacting multiple banks with a loan quote request is a tedious task, loan brokers offer this service to consumers. A loan broker is typically not affiliated with any one bank but has access to many lending institutions. The broker gathers the customer data once and contacts the credit agency to obtain the customer's credit history. Based on the credit score

and history, the broker presents the request to a number of banks that are best suited to meet the customer's criteria. The broker gathers the resulting quotes from the banks and selects the best offer (i.e. the lowest interest rate) to pass back to the consumer" (Hohpe and Woolf, 2004, p.362).

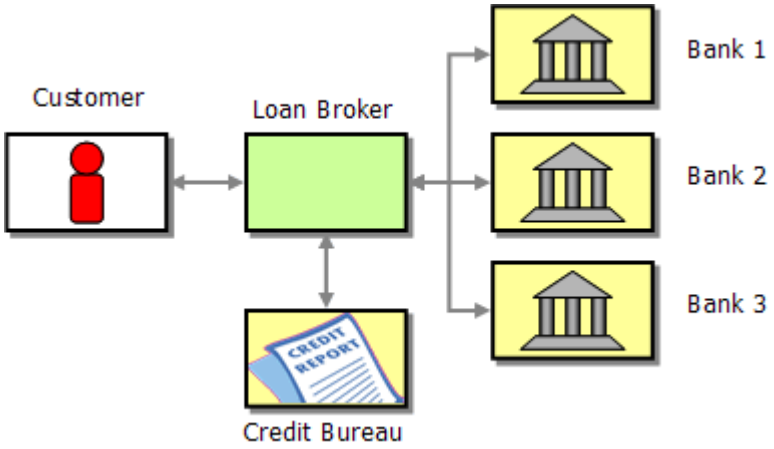

Fig. 5: Simple loan broker (Hohpe and Woolf, 2004)

The four patterns described earlier were deployed for the loan broker scenario. Each deployment used the three different ESBs : Mule, Apache Service Mix, and Oracle Service Bus . For all four patterns, Oracle Service Bus hosted the loan broker application, Mule hosted the credit agency application, and ServiceMix hosted the lender gateway and the bank gateway.

Figure 6 shows the design for the direct connection pattern. Oracle Service Bus had to be connected to the two other ESBs using adapters. The numbers in red correspond to the following ordering of events:

1. Receive the consumer's loan quote request
2. Obtain credit score and history from credit agency
3. Determine the most appropriate banks to contact
4. Send a request to each selected bank
5. Collect responses from each selected bank

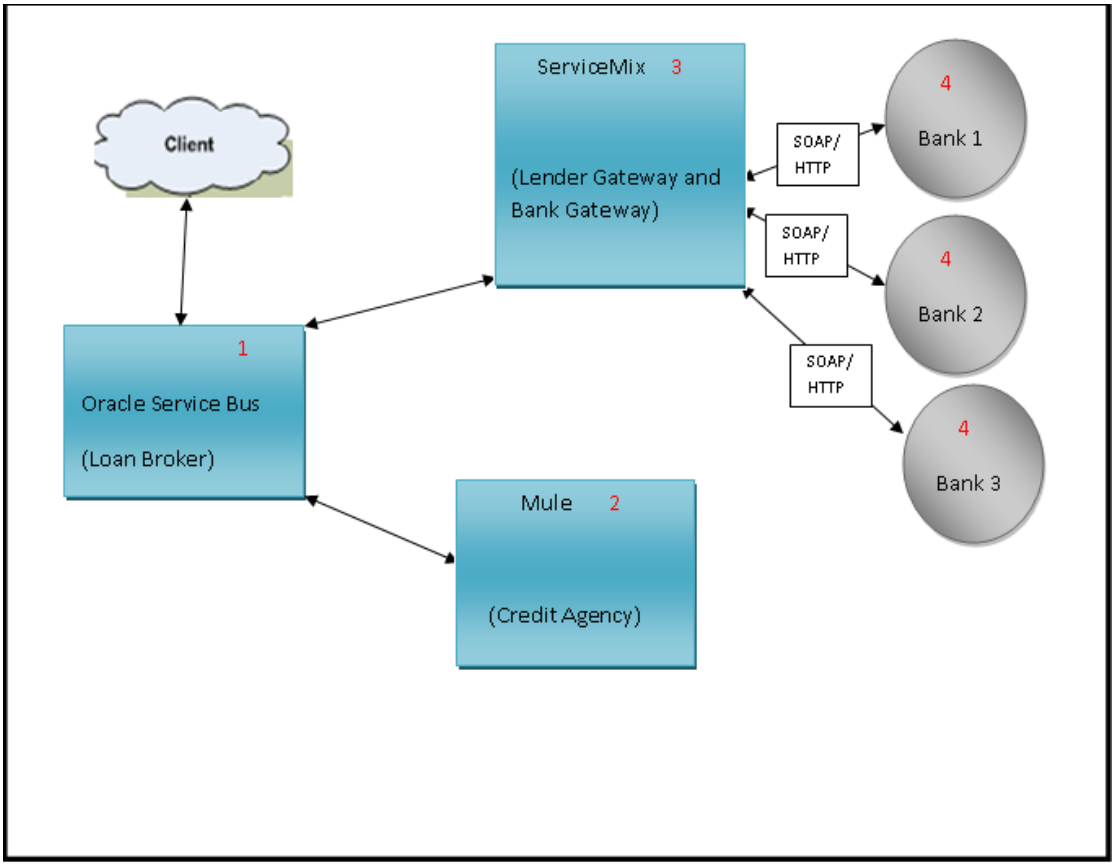

Fig. 6: Direct connection pattern applied to the loan broker scenario

Figure 7 shows the design for the message bridge pattern. An Oracle Web Logic server is used to bridge the ESBs. It was chosen because it is JMS 1.2 compliant, supports the JCA, a standard container for integrating applications, and has an easy-to-use user interface for managing the bridges. JMS messages are used for communication, and the ESBs are interconnected through queues stored in the WebLogic application server. The bank gateway gets quotes from banks based upon the request message and returns the lowest quote to the loan broker via the queue the loan broker is listening to.

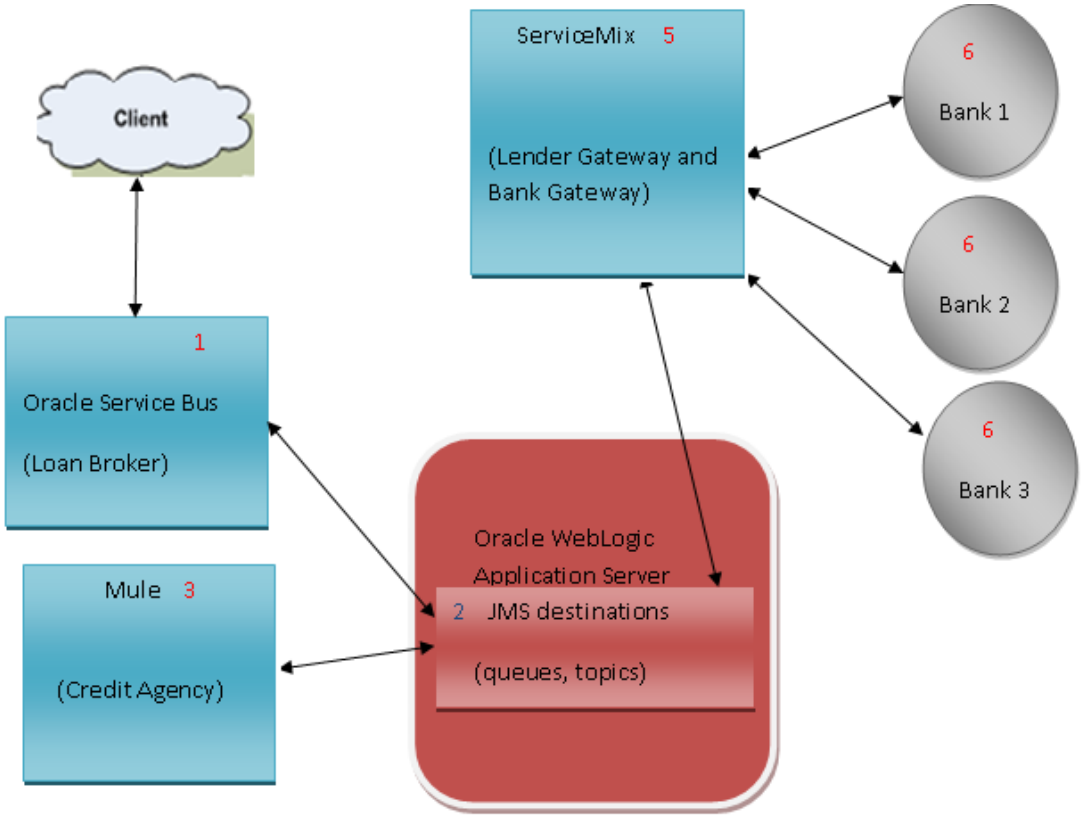

Fig. 7: Message bridge pattern applied to the loan broker scenario

Figure 8 shows the design for the homogeneous messaging middleware pattern. The homogeneous messaging middleware pattern uses IBM WebSphere Message Queue (WMQ) as the JMS implementation, thereby requiring WebSphere MQ client instances installed on the ESBs. “A WebSphere MQ client is a component of the WebSphere MQ product that can be installed on a system on which no queue manager runs.” (IBM, 2005). The WebSphere MQ server is hosted on a different machine from the client. It has a Queue manager that manages the queues (JMS destinations) which play the role of the integration component. Figure 8 shows the network topology for this experiment. It can be seen from the diagram that WebSphere MQ server will host the queue manager. All interactions will be made through the queues managed by the queue manager, i.e. access to the queues is granted by the queue manager to the ESBs when requested.

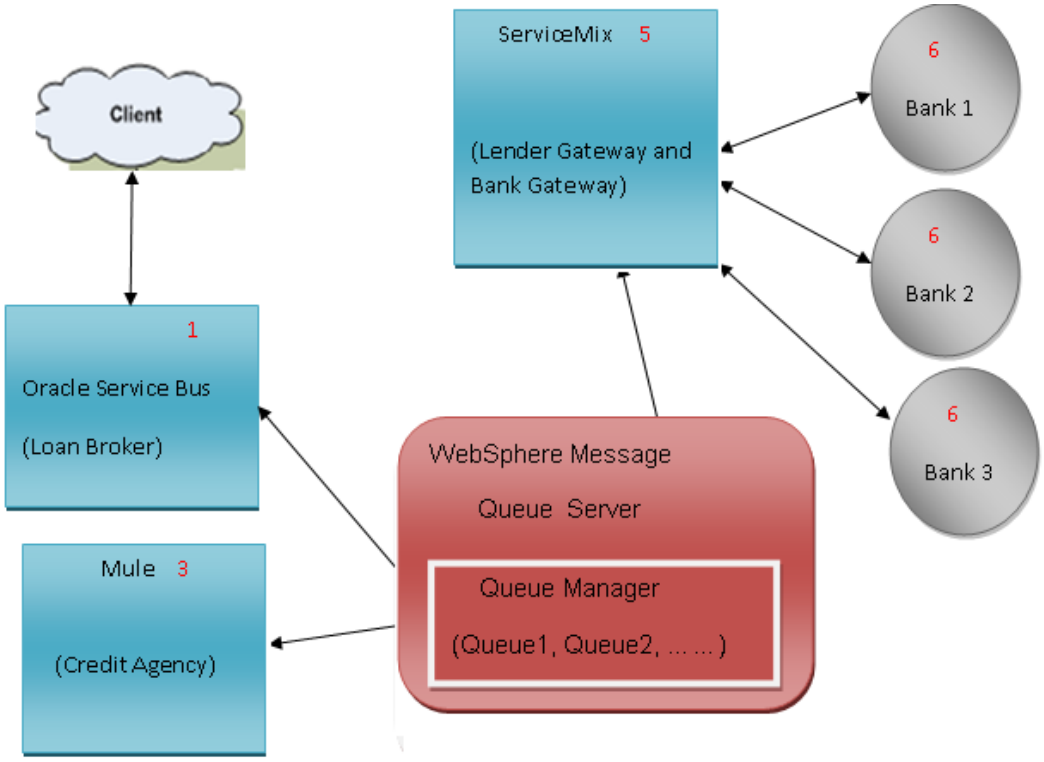

Fig. 8: Homogeneous messaging middleware pattern applied to the loan broker scenario

Figure 9 shows the design for the web services pattern. An Intalio Business Process Management System (BPMS) orchestrates the loan broker process using web services exposed by the ESBs. A final response is sent to the client from the BPMS server, and there is no direct communication between the client application and the ESBs. The process enacted by the BPMS acts as the loan broker, and, as before, Oracle Service Bus hosts the

lender service, ServiceMix hosts the bank gateway, and Mule hosts the credit agency service.

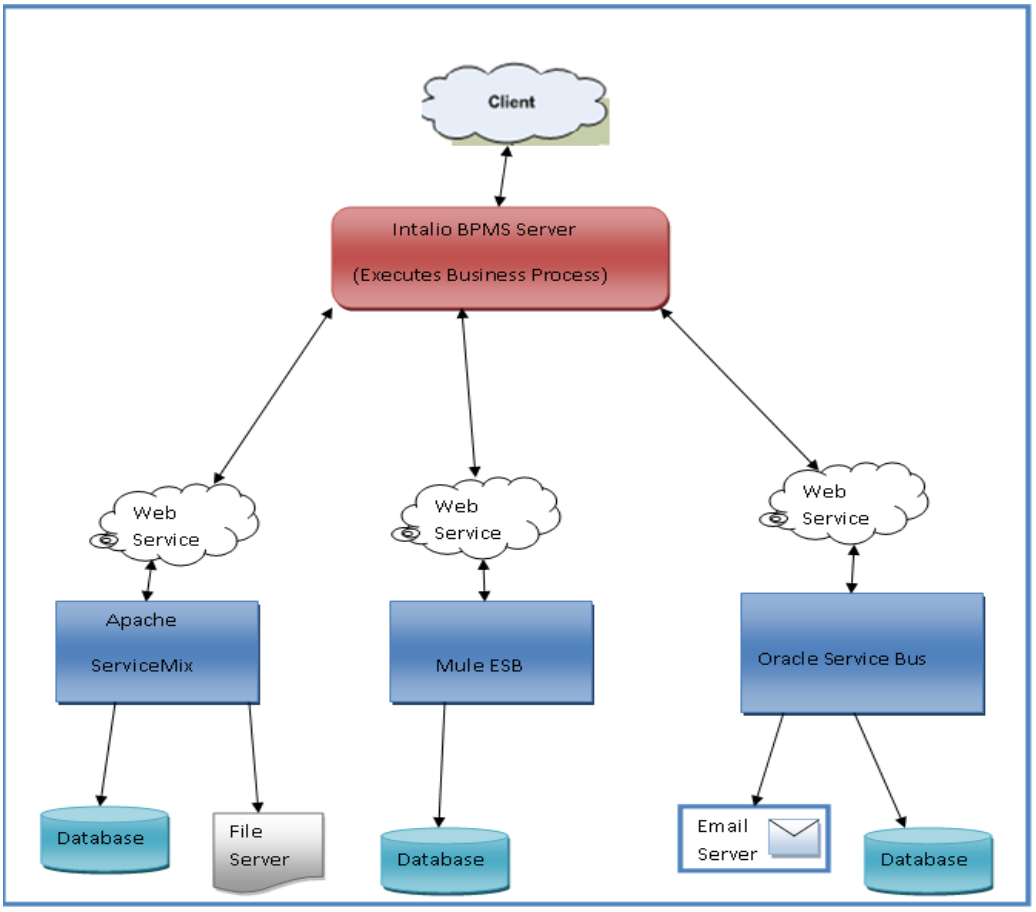

Fig. 9: Web services pattern applied to the loan broker scenario

5. Experiments

In order to test the hypothesis that disparate ESBs can be integrated effectively and with minimal difficulty, each integration pattern was used on each scenario to integrate disparate ESBs. Table 1 below identifies the eight experiments that were performed.

Experiment number	Scenario	Connection pattern	Protocol	Message format	Integrating component
1	Loan Broker	Direct connection	HTTP	XML	None
2	Loan Broker	Homogeneous messaging middleware	WebSphere Message Queue (MQ)	MQ Text message	IBM's WebSphere MQ server
3	Loan Broker	Message Bridge	Java Message Service (JMS)	JMS Text message	Oracle's WebLogic server
4	Loan Broker	Web services	HTTP	SOAP	Intalio's Business Process Management Server
5	Inter-divisional	Direct connection	HTTP	XML	None
6	Inter-divisional	Homogeneous messaging middleware	WebSphere Message Queue (MQ)	MQ Text message	IBM's WebSphere MQ
7	Inter-divisional	Message bridge	Java Messaging Service (JMS)	JMS Text message	Oracle's WebLogic server
8	Inter-divisional	Web services	HTP	SOAP	Intalio's Business Process Management Server

Tab. 1: the eight experiments

The goal of each experiment was to assess the effectiveness and difficulty associated with the integration pattern being tested.

The effectiveness of an integration pattern was assessed as either “Very Effective”, “Effective” or “Not Effective” according to the values of the following variables that were obtained for that pattern in an experiment:

1. Average speed of processing (seconds) : the time taken to receive the response message after the request message has been sent
2. Number of inter-ESB connections
3. Number of possible points-of-failure: each ESB’s connection point (endpoint) is a potential point-of-failure.
4. Suitability for a larger domain (“suitable” or “not suitable”): an integration pattern is deemed “suitable” if it would not increase maintenance costs exponentially, and the number of connections to create would be less than or equal to the number of ESBs added, and it would be easy to manage.

The difficulty associated with implementing an integration pattern was assessed as either “Very Difficult”, “Difficult” or “Easy” according to the values of the following variables that were obtained for that pattern in an experiment:

1. Duration of experiment (hours): the time needed to integrate the ESBs and to test the integration
2. Number of problems encountered: this does not include mistakes made by the implementer
3. Number of services/applications needed: the total includes Java applications, web services and processes

In order to derive scales to be used in the experiments, trials were carried out. The experiment environment was set up and messages were sent between ESBs. The flow of messages was not based on any scenario or business logic. For both scenarios, the required values of variables used as criteria to assess the effectiveness and difficulty was established pragmatically after running the trials as follows.

For the loan broker scenario, for effectiveness, a pattern was deemed “Very Effective” if at least two of the following conditions were met:

1. Average speed of processing <= 1
2. Number of inter-ESB connections = 0
3. Number of possible points-of-failure < 4
4. Suitability for a larger domain: “suitable”

It was deemed “Effective” if at least two of the following conditions were met:

1. Average speed of processing (>1 & <= 3)
2. Number of inter-ESB connections < 2
3. Number of possible points-of-failure = 4
4. Suitability for a larger domain: “suitable”

And it was deemed “Not Effective” if at least two of the following conditions were met:

1. Average speed of processing > 3
2. Number of inter-ESB connections > 1
3. Number of possible points-of-failure > 4
4. Suitability for a larger domain: “unsuitable”

Similarly, for difficulty, a pattern was deemed “Very Difficult” if at least two of the following conditions were met:

1. Duration of experiment > 16
2. Number of problems encountered > 2
3. Number of services/applications needed > 9

It was deemed “Difficult” ” if at least two of the following conditions were met:

1. Duration of experiment (>= 11 & <= 16)
2. Number of problems encountered < 3
3. Number of services/applications needed (> 6 & < 10)

And it was deemed “Easy” at least two of the following conditions were met:

1. Duration of experiment <= 10
2. Number of problems encountered = 0
3. Number of services/applications needed <= 6

The details of the corresponding criteria for the inter-divisional communication scenario are located in (Nwakacha, 2011).

In order to carry out the experiments, four machines needed to be networked together and pre-installed with a variety of software as follows. A dual-processor server with 4GB of RAM and 250 GB of disk space, running Windows 2003, was used. Onto this were installed IBM Web Sphere Message Queue, Microsoft SQL Server 2005 Express Edition, Oracle WebLogic Server and Apache Tomcat. A table was created on the SQL Server to hold information used in the experiments.

Three desktop machines, each running a different operating system (Windows XP, Windows Vista or Windows 7) with 3 GB of RAM and 50 GB of free disk space, were used to host the three ESBs, Mule, Apache ServiceMix and Oracle Service Bus, which were installed on them, one ESB on each machine.

The lender gateway service and bank gateway service were hosted by ServiceMix on one machine and the credit agency service was hosted on Mule on another machine. On the third machine the Oracle Service Bus was used to orchestrate the interactions between all ESBs for all of the experiments except for the two involving the web services pattern. For these, Intalio BPMS Designer and Server products were installed on the same machine as the Oracle Service Bus and used to orchestrate interaction between the ESBs.

Before the experiments, the applications did not have endpoints that could be used to integrate them via the ESBs that hosted them. The experiments focused on how the endpoints were created and used for integration.

The first experiment focused on how the endpoints were created and used when the loan broker scenario was supported by services hosted by disparate ESBs integrated using the direct connection pattern. Figure 10 illustrates the process model for this experiment.

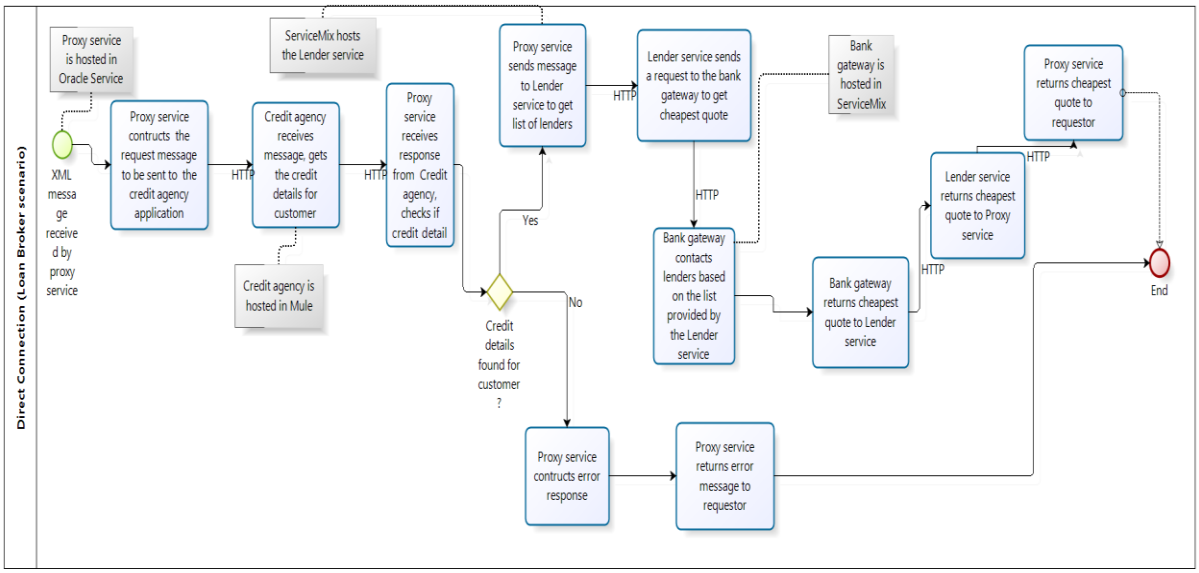

Fig. 10: process model for the loan broker scenario and direct connection pattern

An HTTP endpoint was created on the mule ESB to expose the credit agency service for integration. The lender gateway service and the bank gateway service were hosted on ServiceMix with an HTTP endpoint. OSB was configured for orchestrating the interactions between the services. The details of these configuration activities are recorded in (Nwakacha, 2011, p. 112-119).

One run of this experiment was triggered when the following xml message was sent to the proxy service:

<ln:LoanRequest xmlns:ln="http://www.loanbroker.com/loan">

<ln:Surname>Nwakacha</ln:Surname>

<ln:FirstName>Justin</ln:FirstName>

<ln:LoanAmount>10000</ln:LoanAmount>

<ln:Duration>8</ln:Duration>

</ln:LoanRequest>

Fig. 11: request message for loan broker scenario and direct connection pattern

The experiment ended when the following message was received back by the proxy service:

<ln:LoanResponse xmlns:ln="http://www.loanbroker.com/loan">

<ln:Bank>HSBC</ln:Bank>

<ln:Rate>0.65436</ln:Rate>

<ln:APR>13.6</ln:APR>

<ln:Telephone>08000324735</ln:Telephone>

</ln:LoanResponse>

Fig. 12: response message for the loan broker scenario and direct connection pattern

Figure 13 shows the values for all of the variables for experiment-one. Details of the other seven experiments are recorded in (Nwakacha, 2011, chapter 4). The results of all of the experiments are presented and analysed in the next section.

Experiment: Direct Connection Loan Broker Scenario	
Indicator	**Value / Result**
Duration of experiment	16 hours, 17 minutes
Number of Inter-ESB connections	3
Number of problems encountered	1
Problems encountered 1) The Mule HTTP URL was not accessible by OSB but was accessible from a web browser. **Solution** 1) Changed the version of Mule from 2.2.1 to 2.2.5	
Number of services/applications (Java/WS classes/Processes)	9
Average speed of Processing (Seconds)	1.2
Number of possible points-of-failure	3 (Mule, ServiceMix and Oracle Service Bus)
Ease of implementation (Easy/Difficult/Very Difficult)	Difficult (7-9 services, 1 or 2 problems)
Suitable for larger domain (Suitable/Not Suitable)	Not suitable (met all the conditions)
Efficiency of solution (Not Efficient/Efficient/Very Efficient)	Not Efficient (Not suitable for larger domain, number of inter-ESB connections is greater than 1)

Comments
It is very efficient for 2 ESBs

Fig. 13: results for experiment one: loan broker scenario and direct connection pattern

6. Analysis of results

For the loan broker scenario the results of the experiments for each of the integration patterns is shown in table 2 below. Similarly for the interdivisional messaging scenario, the results are shown in table 3.

	Direct Connection	Homogeneous Messaging middleware	Message Bridge	Web Services
Duration of experiment (hours:minutes)	16:17	15:57	16:30	16:00
Number of problems encountered	1	1	1	0
Number of services/applications needed	9	12	13	7
Difficulty	Difficult	Difficult	Very Difficult	Difficult
Average speed of processing	1.2	2.0	2.4	3.0
Number of inter-ESB connections	3	0	0	0
Number of possible points of failure	3	4	7	4
Suitability for a larger domain	Not Suitable	Suitable	Not Suitable	Suitable
Effectiveness	Not Effective	Effective	Effective	Effective

Tab. 2: Loan broker scenario: results of the experiments with the four integration patterns

	Direct Connection	Homogeneous Messaging middleware	Message Bridge	Web Services
Duration of experiment (hours:minutes)	10:28	9:35	12:15	10:15
Number of problems encountered	0	0	1	0
Number of services/applications needed	9	12	13	7
Difficulty	Difficult	Difficult	Difficult	Difficult
Average speed of processing	1.8	2.0	2.6	3.0
Number of inter-ESB connections	3	0	0	0
Number of possible points of failure	3	4	7	4
Suitability for a larger domain	Not Suitable	Suitable	Not Suitable	Suitable
Effectiveness	Not Effective	Effective	Effective	Effective

Tab. 3: Interdivisional messaging scenario: results of the experiments with the four integration patterns

The tables show that similar results were obtained for both scenarios. The differences in the duration of the experiments for the different scenarios are due both to the different implementation procedures required for each scenario, and to the increasing experience of the experimenter with implementation as the work progressed from the loan broker to the interdivisional messaging scenario. The difference in the average processing speeds between the two scenarios is due to their different message flows.

The results indicate that the Homogeneous Messaging Middleware pattern and the Web Services pattern seem to be the best patterns for integrating disparate ESBs effectively and with minimal difficulty. For both scenarios, the Direct Connection pattern was found to be Not Effective. And, for the loan broker scenario at least, the Message Bridge pattern was found to be Very Difficult to implement.

7. Conclusion

Large companies are increasingly using a set of disparate ESBs in order to support comprehensive enterprise application integration. But the most effective method for integrating such disparate ESBs is not clear currently. This paper has presented the results of

experiments that attempt to assess the extent to which each of four integration patterns imported from the world of EAI could integrate disparate ESBs effectively and with minimal difficulty.

The key finding is that both the Homogeneous Messaging Middleware and Web Services patterns seem to be the best ones to use for integrating disparate ESBs. This result should be useful both to researches and to practitioners in this field.

However, the significance of the results should be tempered with the knowledge of the limitations of the research. First, in regard to the variables, it is not clear that the most appropriate set for assessing effectiveness and ease of implementation has been chosen. And it is also not clear how important each of the chosen variables' contribution is in impacting upon assessments of either effectiveness or difficulty of implementation. Second, with respect to the ESBs, the experiments have all been performed with just three ESBs, so it is not clear whether the trend of the results would still pertain for a larger number of ESBs. In addition, three particular ESBs have been used, and it is not clear that the same results would be obtained if ESBs from other proprietors were to be used. And this is true also of the other software products that were used to create the experiment environment infrastructure. Third, the results obtained will clearly vary with regard to the relative skill of the implementer. However, although the actual values obtained might vary, one could reasonably expect the trend of the results to be as obtained here.

But, despite these limitations, the work does provide a first step in determining the best way to integrate disparate EBSs and, at the very least, is useful for identifying future work that could be undertaken to reach this goal. Such future work includes the following tasks.

1. Use different ESBs (other than the ones used in this research) to carry out the same experiments. This may help to validate the work carried out here.
2. Increase to four or more the number of disparate ESBs being integrated.
3. Use different software applications for the experiment environment infrastructure.
4. Look for other existing or new integration patterns and perform experiments using the discovered patterns.
5. Use different scenarios. This may help support the contention that the results of the experiments are scenario-independent.
6. Investigate other variables, e.g. computer / network resource utilisation.
7. In addition to effectiveness, investigate the efficiency of the integration patterns.
8. The security implications of using the patterns were not covered in this research. But when ESBs reside in different companies (for a virtual network) or in different divisions, for example, security will become an issue. So the impact on security considerations should also be investigated for different patterns.

References

Alonso, G., Casati, F., Kuno, H. and Machiraju, V. 2004. *Web Services: Concepts, Architectures and Applications*, Springer-Verlag, Berlin

Chappell, D. 2004. *Enterprise Service Bus*, O'Reilly, Sebastopol

Hohpe, G. and Woolfe, B. 2004. *Enterprise Integration Patterns*, Boston, Pearson Education.

Keen, M., Bond, J., Denman, J. and Husek, S. 2005. *Patterns: Integrating Enterprise Service Busses in a Service Oriented Architecture*, IBM RedBooks.

Moradi, H. and Bahreininejad, A. 2013. Towards a Comprehensive Framework for Evaluating the Core Integration Features of Enterprise Integration Middleware Technologies, *Journal of Systems Integration*, 4(1), pp 13-29

Nott and Stockton, 2006. Choose an ESB Topology to fit your Business Model, developerWorks, IBM, http://www.ibm.com/developerworks/webservices/library/ws-soa-esbtop/ [accessed online 17th May, 2012]

Nwakacha, J. 2011. Integrating disparate ESBs, MPhil, University of the West of England

Papazoglou, M. and Ribbers, P. 2006. *e-Business: Organizational and Technical Foundations*, John Wiley, Chichester, England

Papazoglou, M. 2012. *Web Services & SOA: Principles and Technology*, 2nd ed., Pearson, England

Powell, 2008. What is an ESB and which do you need? IBM http://www.websphereusergroup.org.uk/wug/files/presentations/24/ESB%20Decision%20Guide%20121107%20AWP.pdf accessed May 17th 2012

Rademakers, T. and Dirksen, J. 2009. *Open Source ESBs in Action*, Greenwich (USA), Manning Publications Ltd.

Schulte, R. 2007. Succeeding with multiple SOA service domains and disparate ESBs, Stamford: Gartner available at http://old.gartner.com/DisplayDocument?ref=g_search&id=504649, [Accessed July 12th, 2012]

Vollmer, K. 2011 The Forrester Wave: Enterprise Service Bus, Q2 2011, Forrester Research, http://www.oracle.com/us/corporate/analystreports/forrester-wave-esb-q2-2011-395900.pdf, [Accessed 16th July, 2012]