COMPUTING SCIENCE

Enterprise Service Bus: An overview

P. de Leusse, P. Periorellis, P. Watson

TECHNICAL REPORT SERIES

No. CS-TR-1037 July, 2007

TECHNICAL REPORT SERIES

No. CS-TR-1037 July, 2007

Enterprise Service Bus: An overview

Pierre de Leusse, Panos Periorellis, Paul Watson.

Abstract

Currently, business requirements for rapid operational efficiency, customer responsiveness as well as rapid adaptability are driving the need for ever increasing communication and integration capabilities of the software assets. Enterprise Application Integration (EAI), which is the process of integrating enterprise systems with existing applications and in general distributed computing, have produced diverse integration techniques and approaches to undertake these challenges. This has brought the development of Service-Oriented Architecture (SOA) variants, which is partly supported by commonly accepted standards that ensure interoperability, sharing and reusability. As a result of this, a safer and faster level of return on investment (ROI) can be generated while inter-software communication and integration has becomes ever easier. In this paper we discuss ESB and evaluate the concept against already existing broker architectures and paradigms.

Bibliographical details

DE LEUSSE, P., PERIORELLIS, P., WATSON, P..

Enterprise Service Bus: An overview
[By] P. de Leusse, P. Periorellis, P. Watson.

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Science, 2007.

(University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. CS-TR-1037)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE
Computing Science. Technical Report Series. CS-TR-1037

Abstract

Currently, business requirements for rapid operational efficiency, customer responsiveness as well as rapid adaptability are driving the need for ever increasing communication and integration capabilities of the software assets. Enterprise Application Integration (EAI), which is the process of integrating enterprise systems with existing applications and in general distributed computing, have produced diverse integration techniques and approaches to undertake these challenges. This has brought the development of Service-Oriented Architecture (SOA) variants, which is partly supported by commonly accepted standards that ensure interoperability, sharing and reusability. As a result of this, a safer and faster level of return on investment (ROI) can be generated while inter-software communication and integration has becomes ever easier. In this paper we discuss ESB and evaluate the concept against already existing broker architectures and paradigms.

About the author

Pierre de Leusse received a MSc in Computing Sciences from the University of TEESSIDE in 2005. He then worked at this University for a year as a researcher/consultant, principally working on automatic discovery of WSs and SOA based knowledge transfer projects between the University and a local SME. He started a PhD is November 2006 under the supervision of Prof. Paul Watson and Dr. Panayiotis Periorellis. His work looks at developing a secure infrastructure for rapid composition of virtual organisations.

Panos Periorellis joined the department in June 2000 as a research associate after successfully completing his Ph.D. in the area of Enterprise Modelling under the supervision of Prof. John Dobson. Panos was promoted to Senior research Associate in March 2004 and started work on the GOLD project looking into issues of trust, privacy and security.

Paul Watson is Professor of Computer Science and Director of the North East Regional e-Science Centre. In total, he has over thirty refereed publications, and three patents. Professor Watson is a Chartered Engineer, a Fellow of the British Computer Society, and a member of the UK Computing Research Committee.

Suggested keywords

SOA,
WEB SERVICES,
ESB

Enterprise Service Bus: An overview

Pierre de Leusse, Panos Periorellis, Paul Watson
pierre.de-leusse@ncl.ac.uk
University of Newcastle
School of Computing Science

Abstract

Currently, business requirements for rapid operational efficiency, customer responsiveness as well as rapid adaptability are driving the need for ever increasing communication and integration capabilities of the software assets. Enterprise Application Integration (EAI), which is the process of integrating enterprise systems with existing applications [1] and in general distributed computing, have produced diverse integration techniques and approaches to undertake these challenges. This has brought the development of Service-Oriented Architecture (SOA) variants, which is partly supported by commonly accepted standards that ensure interoperability, sharing and reusability. As a result of this, a safer and faster level of return on investment (ROI) can be generated while inter-software communication and integration has becomes ever easier. In this paper we discuss ESB and evaluate the concept against already existing broker architectures and paradigms.

Introduction

While from a purely theoretical point of view cheap software integration seems ideal, different aspects of these new technologies and the frequency in which they can be used have generated a more mixed situation. In particular the amount of platforms implemented (.NET, Java, Axis, WebSphere and others) as well as the various specifications related to the different issues associated with EAI or SOA (W3C, OASIS, WS-I) have rendered the middleware environment more opaque. This situation, linked with the raise of inter-application communications [2] and the possible repeated changes of partnerships in these interactions [3], have caused a situation where ROI and ease of integration are difficult to reach.

The traditional opportunistic integration is generally achieved using conventional application-to-application or point-to-point communication [4]. But these approaches have their limits. The complexity grows exponentially with the number of applications or points and the frequency they change. Furthermore, the maintenance and integration costs increase as the application becomes more complex.

Therefore, one objective of more recent integration approaches is to reduce the complexity of integration by replacing the point-to-point ad-hoc with systematic integration through a specialised integration platform. In this paper we present the concept of Enterprise Service Bus (ESB) which is a SOA based software infrastructure that acts as an intermediary layer of middleware through which distributed services and information can be made available.

Evolution of Middleware

The first wave of integration practices aimed to provide APIs and interfaces between systems. As shown on figure 1, this was mostly achieved either using custom Remote Procedure Calls (RPC) or messaging technologies like CORBA. This generation of middleware primarily aimed to achieve point to point integration and most of the connectors were custom built. This generally allowed heterogeneous systems to communicate and in the case of messaging technologies, allowed to store and to forward messages. On the other hand, specific interfaces had to be developed for every system involved but at the same time the lack of widely spread communication semantics meant that many different formats prospered, thus generating a low level of reusability and rendering all coding and maintenance complex.

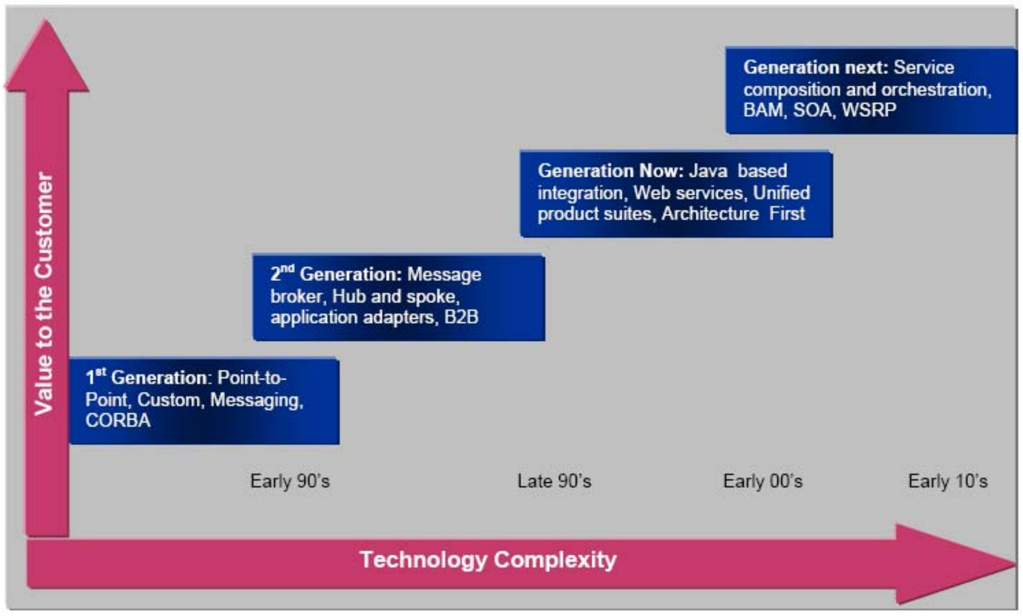
Generation next: Service composition and orchestration, BAM, SOA, WSRP
Generation Now: Java based integration, Web services, Unified product suites, Architecture First
2nd Generation: Message broker, Hub and spoke, application adapters, B2B
1st Generation: Point-to-Point, Custom, Messaging, CORBA
Value to the Customer
Early 90's
Late 90's
Early 00's
Early 10's
Technology Complexity

Figure 1 Integration history [4]

The next generation mainly aimed to improve the reusability as well as connectivity issues and introduced the spoke-hub distribution paradigm. Common integration infrastructures subsequently developed include, Application Servers (AS) and EAI brokers such as hub-and-spoke architectures that can potentially offer features such as message routing, transformation, business rules enforcement, transaction monitoring and auditing. This approach allowed reducing the connection complexity as with n nodes, a maximum of n - 1 routes are necessary to connect all nodes, compared to (n(n - 1))/2 nodes that would be required in a point-to-point network. In addition, this made possible to re-use and share the integration logic among multiple projects. These advantages provided connectors that were potentially faster to put in place and easier to maintain. However, the broker or AS due to its hub-and-spoke nature can create a bottleneck effect, impacting on the performance as well as creating a single point of failure. Furthermore, although this type of infrastructure can support exchange format standards, this aspect was still being neglected during this age. Finally, this type of middleware links the connected systems together in a tightly coupled fashion, as it often intertwines the application and the integration logic.

With inter-application integration passing from a consideration to one of the main centres of interest, the current generation of middleware focuses on loosening the coupling and developing common exchange semantics. This brought up the emerging growth of the SOA paradigm and technologies such as REST or Web Services (WS) and Message Oriented Middleware (MOM) in particular. With this concept, distributed systems can rely upon independent services in which the application specific logic is independent from the connection infrastructure. In addition, with the adoption of XML and the large effort put into defining common specifications and standards, more flexibility has been granted. Both software layers segregation and semantic rationalisation allow the creation of applications that are built by combining loosely coupled and interoperable end points. Services are widely used to abstract different application specific logics to reflect business activities [5]. These activities can be reused and combined to compose new services or processes [6] without impacting the underlying activities. This is generally acknowledged as being the main commercial interest of SOA [5, 7] as it allows a faster and safer ROI. But one major factor that slows down the development of efficient process composition is the fair amount of hard coding still required by current middleware infrastructures to be connected to each other. Furthermore, the extensive use of connecting infrastructures, their complexity and the increasing need to connect them are rendering the middleware landscape more complex.

We do acknowledge that SOA for some people is simply a marketing term for the packaging of the communication infrastructure, which in actual fact should not matter. Interoperability and connectivity issues have been discussed for the past 20 years, however what SOA has brought about is the need to integrate at the middleware level using standardised technologies.

It is indeed interesting to note that while SOA eases the diversity and heterogeneity issues inherent to distributed systems; it does not completely solve them. In fact, the present standardised ways to abstract and simplify application specific logic used by middleware to address inter-software communication issues can also cause problems between middleware structures. These issues are intrinsic to heterogeneous environments where different interests and practices meet. The next generation of middleware should respond to this challenge and alleviate inter-middleware communications to allow more dynamic service and process compositions, thus enabling more effective business collaboration. This is suggested in the SOA Maturity model [8] which advocates that top level SOA infrastructures should be able to more dynamically adapt to changes.

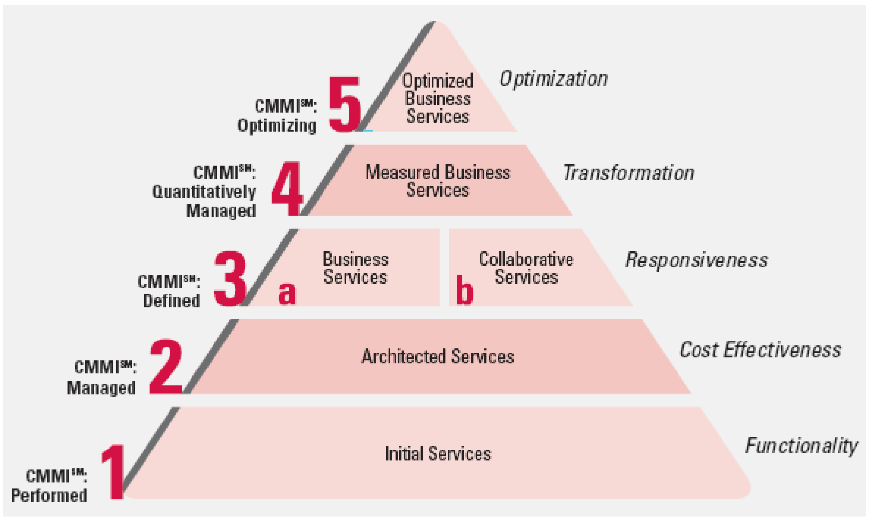

Figure 2 A Service-Oriented Architecture Maturity Model

Enterprise Service Bus

An answer to potentially enhance the adaptability of middleware has been found in leveraging the infrastructures and practices produced during the evolution of integration software previously introduced.

The hub-and-spoke architecture has the benefit of being centralised, which provides a good structure for key features such as message routing or audit as well as allows a higher level of reusability. However it does not scale well across heterogeneous and large distributed systems due to the centralised nature of the middleware itself. In addition, the historical lack of common exchange semantics hinders the creation of evolving collaborations. These issues were partially solved by SOA, but while allowing a more loosely coupled model, it requires time consuming low level coding. Moreover in a highly evolving environment, MOM necessitates a higher level of abstraction to allow the reusability and integration levels necessary for a fast ROI.

Figure 3 shows some of the higher level qualities of these infrastructures and introduces the concept of ESB. The ESB architecture applies knowledge learnt throughout the evolution of middleware and attempts to leverage the technologies subsequently developed [9]. Indeed the bus takes the centralised approached of the AS, the abstracted nature of the EAI broker with the distributed nature of MOM to provide a solution for rapidly adaptable middleware.

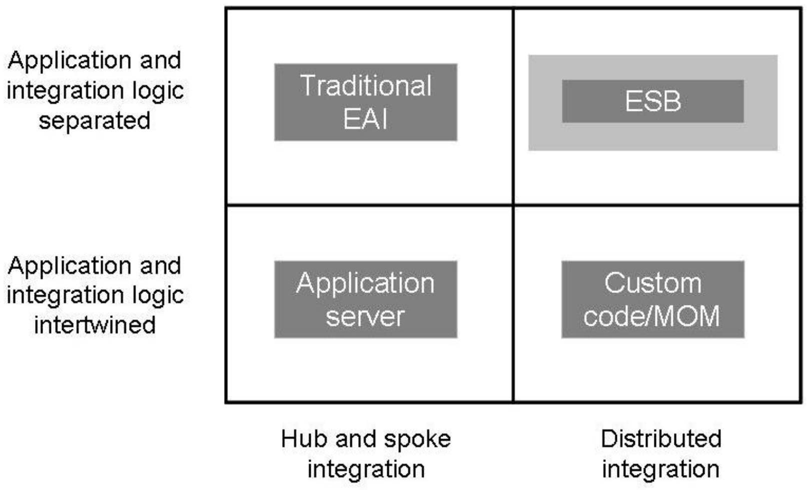
Application and integration logic separated
Traditional EAI
ESB
Application and integration logic intertwined
Application server
Custom code/MOM
Hub and spoke integration
Distributed integration

Figure 3 Integration approaches [10]

Definition

Although the exact definition of ESB varies according to author, company or features it includes, it is possible to draw a general picture.

An ESB is a SOA based software infrastructure that acts as an intermediary layer of middleware through which distributed end points are made available. It provides an abstraction layer that acts as entry point to a bus. Once the messages have been intercepted by the entry point, series of actions can take place in the bus. These actions take the form of services which are called according to various elements such as: message content, origin, destination and sets of rules predefined.

"*The pattern of an ESB is that it is a single shared communications framework for all service interactions to pass through. And once that happens it can do the mediation, logging, and routing that is required.*" Paul Fremantle [11]. Ideally we would like to add to this list of potential features and examine the feasibility of using ESBs to implement non-repudiation, message level security, authentication components such as an identity provider or authorisation components such as a policy decision point. The purpose of this investigation into ESB is to examine the extent which they can be used to alleviate individual services of the burden of implementing or deploying additional code in the form of handlers to deal with the above features.

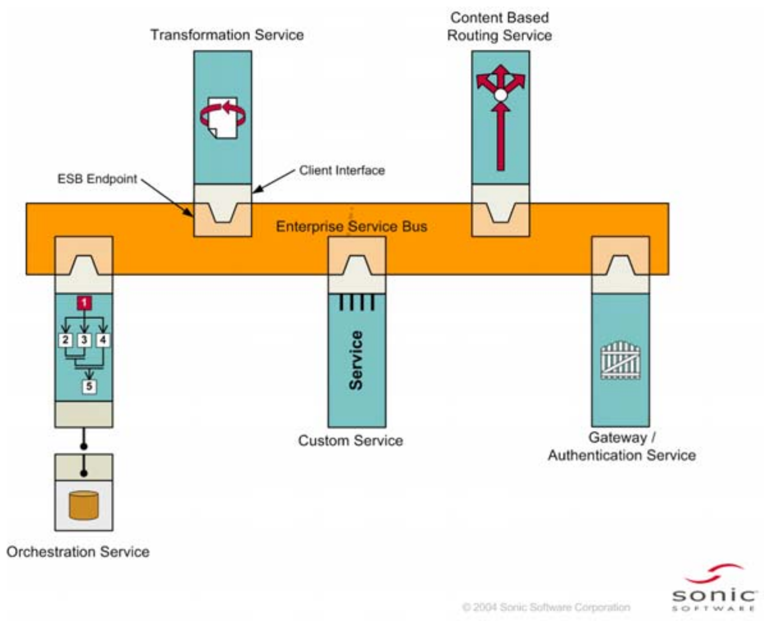

Figure 4 Generic ESB example

Figure 3 shows a basic ESB infrastructure in which services can communicate through an abstracted interface provided by the bus. Once a message is received by the bus end point, series of actions can take place. The type of actions taking place is influenced by both the content of the messages received and the way the bus has been configured to handle them. In this specific example the bus offers authentication, transformation, content base routing and orchestration services. Finally, once the messages have been treated it is sent to the appropriate end point with the suitable data.

Characteristics

The characteristics are separated in two categories: core and extended. The core characteristics include the features generally offered by ESB infrastructures while the extended attributes present optional facets that can be interesting additions to a bus.

Core

The architectural pattern of ESB relies on five main components that are introduced below .The first important characteristic of an ESB is that it is an implementation of SOA. Like other SOA based structures its components can be distributed across a network for the purposes of performance, quality of service, industrial agreements and economics. As a product of the SOA paradigm it mostly uses XML as the standard communication language. Other alternatives can be found in CSV (Comma-Separated Values), EDI or more recently developed YAML [12] and JSON [13].

In order to fulfil their role as universal mediators between distributed systems, ESBs generally aim to support web services standards. This is well described by the WS-I basic profile [14, 15] which starts with XML and goes through SOAP and Web Services Description Language (WSDL).

After underlining the importance of its structure, we must put the emphasis on what it delivers. As seen above, the core functions of an ESB are routing, invocation, and mediation.

Routing deals with addressability and content or metadata based redirections. In order to be able to identify and use the services, brokers and other components that constitute the ESB, it is necessary to be able to specify the workflow linking these components. It is often called intelligent routing because it can be made able to take decisions on this workflow based on the results of a previous operation. The WS-Addressing [16] specification is the most widely use at this effect. WS-Addressing defines mechanisms for asynchronous messaging, can contain information about WS endpoints and allows a call-back to be performed.

Invocation is about the capacity to make requests and receive responses. The ESB must provide some form of connection pipe that act as a bus through which all exchanges can be done. Web service implementations have widely different expectations of time to completion; therefore the bus has to support asynchronous messaging.

Finally, mediation refers to the capacity to transform messages or part of messages between different formats and standards. It includes transformation services between the format of the sending application and the receiving application to facilitate the transformation of data formats and values. Multiple transformations can follow each other to implement complex transformation logic. The use of XML being common, the transformation is often performed with XSLT or XQuery [17].

Leveraging existing standards

Given the current rise in WS standards it is believed that ESBs –as a conceptual construct- offer the ideal application domain to leverage and experiment with current WS specifications. In the case of messaging we would like to leverage the existing WS Security stack such that SOAP messages can be used to guarantee certain security properties. In particular we would like to structure messages according to the WS Security specification so that we can provide a controlled mechanism for the various properties we want to guarantee.

We are exploiting technologies such as X509 certificates as well as SAML tokens as potential technologies for message authentication. Depending on the type of authentication architecture adopted (with single sign being the most likely) we are exploring ways of combining WS security with Liberty alliance technologies such as the security assertions offered in the SAML specification.

Since security tokens are specific to the service, with the service provider being in charge of the tokens he accepts, WS Policy and its related specifications for security and metadata attachments can be used to allow distributed services to express, as part of their WSDL descriptions, specific security issues (or policies) regarding their services. The WS Policy attachment specification allows a WS Policy to be referenced via a WSDL interface so that the service consumer can retrieve and make use of that policy. It is envisaged that such policies can be part of and maintained by our ESB infrastructure. This will provide additional flexibility to the service consumer while at the same time service coupling can be further loosened. It is envisaged that service consumers will be able to process metadata (related to security) on the fly before preparing a message.

In addition WS addressing can be used as a routing protocol to partly ensure point to point security (we say here partly because XML encryption and XML Signature enable point to point public key cryptography). SOAP message headers can be used to inform the recipient (be it a service) where an asynchronous message should be forwarded to, after it has been processed. WS Security structured headers provide specific placeholders for WS addressing assertions so that the sender of an asynchronous communiqué can supply the endpoint and other details of the destination of the response. Additional public key cryptography as we mentioned earlier guarantees confidentiality and integrity between the sender and the receiver of a message.

Authorisation standards such as XACML provide a standardised protocol for requesting access to a resource and providing feedback to that request. In addition it enables messages to include such requests as part of their WS security headers. It is envisaged that ESBs can also play the role of the authorisation authority as a typical 3rd trusted party. Policies can be logically centralised as part of the ESB or decentralised and available on the fly as and when an ESB requests them. ESBs can be flexible by simply playing the role of the policy decision point only or exercise the rights of a policy holder and a policy maker although we should stress these are strictly business level decisions.

WS Eventing and WS notification are investigated as potential protocols for enabling ESBs to act as a notification broker. Typically in a workflow scenario where several services participate in achieving an objective, various services are interested for various events. These events usually trigger further events within a workflow scenario and the best way to inform of an event taking place is to provide a notification broker whose job is to send messages to various interested parties when an event has taken place. We are exploring the possibility of using ESBs also as notification broker and leveraging existing WS standards and their corresponding mechanisms that dictate how a notification should be sent, how interested parties should subscribe for notices what interfaces should be implemented etc. Since some of the communications via the ESB are bound to be of high importance we are looking into potential protocols that would allow us to maintain irrefutable evidence of certain messages being exchanged.

Extended

In addition to the core features introduced above, an ESB could benefit from using some of the key SOA infrastructure components. These are some of the services we believe that could enhance the functionality of ESB’s.

A service registry can be used to provide a central store of service definitions. Services are published onto the registry which makes them available. Universal Description, Discovery and Integration (UDDI) [18] is one of the core Web services standards. It provides access to WSDL documents and other metadata describing the protocol bindings and message formats required to interact with the web services listed in its directory.

Regulation services can be put in place to provide the monitoring and control mechanism to apply security and business policies inherent to multi-organisational distributed systems.

On the top of the core invocation feature introduced above, ESBs can provide more advanced types of messaging such as synchronous, publish/subscribe, store and forward (queuing).

Synchronous messaging, involves sending a message and waiting for a reply before proceeding and this can be used to make sure the service partners have processed certain messages or operations. The publish/subscribe model allows authorised subscribers to receive information when it is published. The WS-Notification [19] and WS-Eventing [20] specifications are two attempts to answer to this demand. Store-and-forward allows holding messages in contexts where variable levels of availability are expected, when an operation requires human action for instance.

Services often stem from different technological and organisational environments. In order to alleviate their composition it is potentially useful to allow the ESB to understand process semantics. Technologies such as BPEL could be used to enable ESB's to enact workflows. WS-BPEL (Business Process Execution Language) is a widely spread business process executable modelling language that aims to allow web services interactions in a process like model.

An integration tool is frequently placed between different organisations and trust domains. This creates concerns related to security and more specifically to control the access to the different components and services. ESB can include a security model to authorize and authenticate users and message sources. A specific issue related to the ESB's distributed architecture and use, is the necessity to differentiate the access rights of its components as opposed to those of the users, these being potentially intertwined. The central place of the ESB in a distributed environment also renders it appropriate for handling, increasing or enforcing security. More specifically, the broker can provide authentication for its users. Furthermore, with an authorisation service it can perform permission checks to see if users have access to certain actions. In addition, message level encryption can be applied following predefined rules. Finally, attestation that messages haven't been modified can be provided with an integrity service.

Reliability is an important aspect of software engineering where ESB can potentially have an influence. The use of a mediator placed between participants potentially leaves the place for message queuing and saving which could avoid the loss of data. In case of software, system, or network failures on one side of the bus, the message stored can be resent through its normal route or by a dedicated canal. Finally, with its central place as a pipe through which all messages pass, the ESB can also rightfully be used to keep an audit track.

Conclusions

In this paper we have presented the architectural pattern of ESB. We discuss how ESBs try to leverage different technologies and designs used throughout the middleware world thus moving away from the opportunistic point to point inter-software communication and adopting the centralised approach already introduced by the traditional EAI broker. In addition we showed how the ESB concept takes advantage of the SOA paradigm benefits which jointly provides a potential architecture for highly distributed and adaptable as well as loosely coupled middleware. Finally, we have briefly introduced our research goals related to the use and potential improvements of these features.

References

1. Losavio, F., D. Ortega, and M. Perez. *Modeling EAI [Enterprise Application Integration]*. in *Computer Science Society, 2002. SCCC 2002. Proceedings. 22nd International Conference of the Chilean*. 2002.
2. Natis, Y.V., et al., *Predicts 2007: SOA Advances*. 2006.
3. T. Dimitriakos, G.L., et al, *Towards a Grid Plateform Enabling Dynamic Virtual Organisations for Business Applications*. 2005.
4. Radhakrishnan, S., *Integrating Entreprise Applications: Backgrounder*. 2005.
5. Erl, T., *Service-Oriented Architecture Concepts, Technology, and Design*. 2005.
6. Peltz, C., *Web services orchestration and choreography*. Computer, 2003. **36**(10): p. 46 - 52.
7. Erradi, A.M., P., *wsBus: QoS-aware Middleware for Reliable Web Services Interactions*. e-Technology, e-Commerce and e-Service, 2005. EEE '05. Proceedings. The 2005 IEEE International Conference on, 2005: p. 634 - 639.
8. Bachman, J., S. Kline, and B. Soni, *A New Service-Oriented Architecture Maturity Model*. 2005.
9. Corporation, S.S., *SONIC ESB: AN ARCHITECTURE AND LIFECYCLE DEFINITION*. 2005.
10. Chappell, D., *Enterprise Service Bus*. 1st ed. Vol. 1. 2004: O'Reilly. 247.
11. *Colorado Software Summit 2005 - Paul Fremantle. http://www.softwaresummit.com/2005/speakers/fremantle.htm*.
12. *YAML, 2006. http://www.yaml.org/*.
13. *Introducing JSON, 2006. http://www.json.org/*.
14. *Web Services Interoperability Org, 2006. http://www.ws-i.org/*.
15. Senthil Kumar, K.M.A.S.D.S.P., *WS-I Basic Profile: a practitioner's view*. IEEE International Conference on Web Services, 2004. Proceedings. , 2004: p. 17 - 24.
16. *Web Services Addressing (WS-Addressing), 2004. http://www.w3.org/Submission/ws-addressing/*.
17. *XQuery 1.0: An XML Query Language, 2007. http://www.w3.org/TR/xquery/*.
18. *Introduction to UDDI: Important Features and Functional Concepts*. 2004.
19. *Web Services Notification (WSN), 2006. http://www.oasis-open.org/committees/wsn/*.
20. *Web Services Eventing (WS-Eventing), 2006. http://www.w3.org/Submission/WS-Eventing/*.