Designing An Enterprise Service Bus (ESB) Architecture As Business Model Protocol (BMP) For Distributed Electronic-Commerce Systems And Applications.

Anibrika S.K. Bright[1], Dr. M. Asante[2], Ashigbi F. Degadzor[3], Mustapha M. Adamu[4]

Koforiuda Polytechnic, Koforidua Polytechnic[1]
Christ Apostolic University College, Kumasi[1]
Kwadaso.
Contact: +233 0506618729, Eastern Region, Ghana, West Africa.[1]
Computer Science Dept., Knust, Kumasi, Ghana, W/A
Contact: +233 208168613, Ashanti Region, Ghana West Africa.[2]
Department Of Computer Science, Koforidua Polytechnic
Departemnt Of Computer Sceince, Koforidua, Ghana[4]
Contact: +233244224737, Eastern Region, Ghana, West Africa.

ABSTRACT

Enterprise Service Bus (ESB) architecture is considered a platform to realize a service-oriented architecture. An ESB brings flow-related patterns such as transformation and routing of messages and applications through a Service-Oriented Architecture platform (SOA). An ESB can also provide an abstraction of layers for endpoints. This promotes flexibility in the transport layer and easy connection and communication between services. This paper therefore seeks to model and design a service oriented architecture that creates a platform for business modules intercommunication that would establish a reliable protocol to enable efficient and secure interaction between modules and look at the feasibility of implementing an electronic-commerce platform based on the Enterprise Service Architecture (ESA). To achieve these objectives, this paper would consider the Enterprise Service Bus architecture as the blueprint that represents the piece of software residing between the business applications and enables communication among them. Ideally, the ESB should be able to replace all direct contacts with the applications on the bus, so that all communication takes place via the ESB. To achieve this objective, the ESB (Enterprise Service Bus) must encapsulate the functionality offered by its component applications in a meaningful way. This typically occurs through the use of an enterprise message model. The message model defines a standard set of messages and protocols that the ESB will both transmit and receive. In an enterprise architecture making use of an ESB, an application will communicate via the bus, which acts as a message broker between applications and platforms. Such an approach has the primary advantage of reducing the number of point-to-point connections required to allow applications to communicate. This, in turn, makes impact analysis for major software changes simpler, modular and more straightforward. By reducing the number of points-of-contact to a particular application, the process of making a system to changes in one of its components becomes easier. In conclusion faster and cheaper communication between existing systems ensure increased reliability that ensure point-to service solutions to enterprise-wide deployment (distributed bus) and predefined ready-for-use service and application types types. On the other hand, there would be increased overhead and slow down communication speed for those already compatible services and applications.

Keywords: *Deployment, services, application, architecture, bus, protocol.*

INTRODUCTION

In software engineering, a Service-Oriented Architecture (SOA) is a set of principles and methodologies for designing and developing software in the form of interoperable services. These services are well-defined business functionalities that are built on software components (discrete pieces of code and/or data structures) that can be reused for different purposes-program modularity. SOA design principles are used during the phases of systems development and integration. One of the main platforms that was introduced is the Enterprise Service Bus referred to as ESB. ESB is considered a platform to realize a service-oriented architecture. An ESB brings flow-related concepts such as transformation and routing to a Service-Oriented Architecture. An ESB can also provide an abstraction for endpoint applications. This promotes flexibility in the transport layer and enables loose coupling and easy connection between services (BEA et al) . The role of the IT architect is to evaluate business problems and build solutions to solve them. The architect begins by gathering input on the problem, developing an outline of the desired solution, and considering any special requirements that need to be factored into that solution. The architect then takes this input and designs the solution, which can include one or more computer applications that address the business problems by supplying the necessary business functions. To improve the processes over time, capture and reuse the experience of the IT (Information Technology) architects in such a way that future engagements can be made simpler and faster. This is done by capturing knowledge gained from each engagement and using it to build a repository of assets (Braga, Rubira and Dhab, 1998). Information Technology (I.T) architects can then build future solutions based on these proven assets. This reuse saves time, money, and effort and helps ensure delivery of a solid, properly architected solution. An important subset of them uses web services and is designed using Service-Oriented Architecture (SOA) principles. A definition SOA as an architectural style in which a system is composed from a set of loosely coupled services that interact with each other by sending messages (packets of data). In order to interoperate, each service publishes its description, which defines its interface and expresses constraints and policies that must be respected in order to interact with it. In this architectural style, applications are built by coordinating and assembling services in the form of a workflow that invokes services as needed, as well as standard software components. A service is a logical representation of a business activity that has a specified outcome (Buschmann and Sommerland, 1996) . A key principle about services is that they should be easily reusable and discoverable, even in an inter-organizational context. Furthermore, the channels of communication between the participating entities in a service-oriented application are much more vulnerable than in operating systems or within the boundaries of an organization’s intranet, since they are established on public networks. The complexity of the software used to handle web services adds to the total complexity and can be a source of attacks, which makes security an important concern (Chappell et al, 2004) .An Enterprise Service Bus (ESB) is an infrastructure component for integrating applications and services. ESBs facilitate the connectivity of business logic, where this business logic is represented as a service. Most businesses have a heterogeneous environment with applications implemented in various application programming models, such as J2EE, .Net, so on. One goal of universal connectivity, of an ESB, is to allow these different applications (services) to be connected (Corsaro et al, 2002). An ESB facilitates this connectivity by providing transformations that allow the invocation of a service or a service request, which is presented in one format to be “transformed” to a different format which the service provider can respond to. ESB may provide support for:

1. Assembling and routing of messages between services
2. Converting transport protocols between requestor and services
3. Transformation of message formats between requestor and services
4. Handling of business events and processes from disparate sources

At a conceptual level, an ESB allows the application designer/developer to build an application framework that invokes services without having to know where these services are located or how they are invoked. An ESB is responsible for the routing of messages between service requestors and service providers (Erl et al, 2009).

CONCEPTUAL FRAMEWORK

Many authors in the past have had extensive research on the Service Oriented Architectures like the Enterprise Service Bus (ESB) that allows the separation of application logic by providing a “Police Layer” that serves as an interface for monitoring

,coordination and management of service requests and response by various application that interact with other for resources.
The following papers were found during the literature review, and passed our selection criteria.Enterprise Service Bus: A Performance Evaluation Published by Sanjay P. Ahuja, Amit Patel in 2011. This paper also becomes useful because it contained a performance analysis between different types of ESBs. SOAs & ESBs Published by J. Paisley in 2005. This paper also brings up some of the challenges that can be encountered when implementing an ESB and also some of the challenges that can be encountered when implementing a complex ESB solution. Service-Oriented Performance Modeling was alsopropounded by the MULE Enterprise Service Bus (ESB) Loan Broker Application, Published by P. Brebner in 2009. This paper handles both scaling and performance in ESBs. An integration strategy for large enterprises also Published by Dejan Rismic in 2006. It provides information on important aspects of ESB. This is neededto be able to understand and improve the ESB. Performance Prediction of Service-Oriented Applications based on an Enterprise Service Bus, Published by Yan Liu, Ian Gorton and Liming Zhu in 2007. This paper handles important aspects of ESB performance and communication protocols that ensure effective and efficient request and response platform.

METHODOLOGICAL FRAMEWORK

In this section, a platform of the Service – Oriented Architecture called the Enterprise Service Bus (ESB) is considered in relation to electronic business modeling. First of all the Enterprise Service Architecture reference is looked at as the basic architecture prototype for electronic commerce modeling.

METHODOLOGICAL OBJECTIVE

Provide a convenient infrastructure to integrate a variety of distributed services and related components in a simple way.

SCENARIO

An Electronic Commerce agency interacts with many services to do flight reservations, check hotel availability, check customer credit, and others. This interaction is being done now by direct interaction, which results in many ad hoc interfaces, and requires many format conversions. The system is not scalable and it is hard to support standards. Distributed applications using web dedicated services, as well as related services such as directories, databases, security, and monitoring.There may be also other types of components (J2EE-Java 2 Enterprise Edition, .NET-any web compliant programming language). There may be different standards applying to specific components and components that do not follow any standards.

METHODOLOGICAL APPROACH

When an organization has many scattered services, how can one aggregate them so they can be used together to assemble applications, at the same time keeping the architectural structure as simple as possible and hidden from the user, and apply uniform standards? The solution to this problem is affected by the following forces and programming indicators:

• Interoperability. It is fundamental for a business unit in an institution to be able to interact with a variety of services, internal or external.

• Simplicity of structure: you want a simple way to interconnect services; this simplifies the work of the integrators.

• Scalability: one need to have the ability to expand the number of interconnected services without making changes to the basic architectural design.

• Message flexibility: one need to also provide a variety of message invocation styles (synchronous and asynchronous) and formatting. you can thus accommodate all component needs.

• Simplicity of management: you need to monitor and manage many services, perform load balancing, logging, routing, format conversion, and filtering.

• Flexibility: New types of services should be accommodated easily.

• Transparency: you should be able to find services without needing to know their locations.

• Quality of service: you may need to provide different degrees of security, reliability, availability, or performance.

• Use of policies: you need a policy-based configuration and management. This allows convenient governance and systematic changes. Policies are high-level guidelines about architectural

or institutional aspects and are important in any system that supports systematic governance .
• Standard interfaces: you need explicit and formal interface designs (e.g common GUI-Graphical User Interface that is user-friendly).

SOLUTION

Introduce a common bus structure that provides basic brokerage functions as well as a set of other appropriate services. Figure 1 shows a typical structure (Morrison and Fernandez, 2006). One can think of this bus as an intermediate layer of processing that can include services to handle problems associated with reliability, scalability, security, and communications disparity
(Morrison and Fernandez, 2006). An ESB is typically part of a Service-Oriented Architecture Implementation Framework, which includes the infrastructure needed to implement a SOA system. This infrastructure may also include support for stateful services. Below is the diagram of the Enterprise Service Bus architecture reference model and the derived architecture that can be used as a suitable platform for modeling and designing electronic commerce systems based on the ESB architecture (MuleSoft et al, 2001).

STRUCTURE

Figure 1(a) shows the class diagram of the ESB pattern. The ESB connects Business Services with each other providing support for the needs of these services through aService Infrastructure made up of Business Application Services (BASs), which in turnuse Internal Services to perform their functions. BASs are accesses through Service Interfaces (SIs) (TradeSoft et al, 2001).

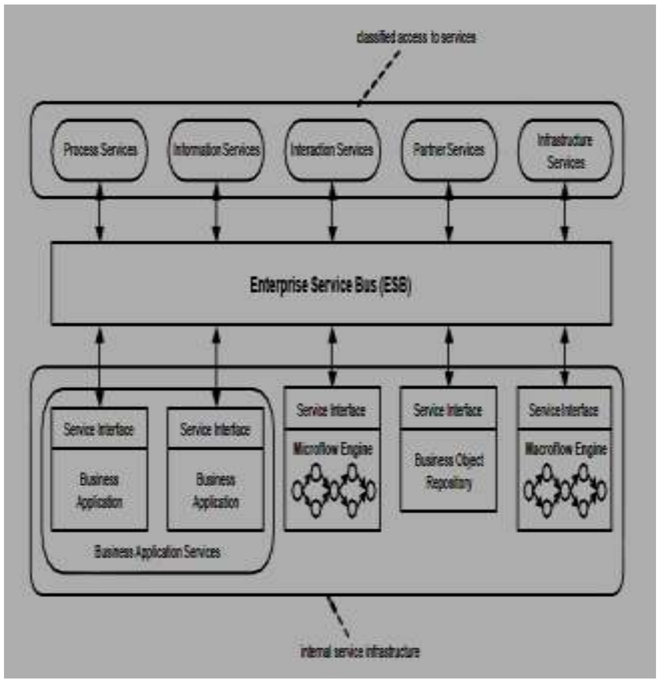

Figure 1 (a) The internal service infrastructure model of (ESB) Architecture (Papazoglou and Heuvel, 2007).

RELATED PATTERNS (COMPONENTS AND INTERFACES)

The ESB is a type of Message Channel Broker and it is also closely related to the Message Bus pattern, both described because of its role as a communicator, the ESB is related to a variety of patterns that provide communication or adaptation. The ESB can be seen also as a microkernel in that it forwards client requests to a set of services (Rouselle et al, 2002). The Enterprise Service Bus can be considered a composite pattern comprised of the following patterns:
• The (Service) Broker pattern which itself is a composite pattern that consists of a set of integration-centric patterns used to translate between incompatible data models, data formats, and communication protocols.
• Asynchronous Queuing pattern which establishes an intermediate queuing mechanism that enables asynchronous message exchanges and increases the reliability of message transmissions when service availability is uncertain.
• Intermediate Routing pattern which provides intelligent agent-based routing options to facilitate various runtime conditions (Rouselle et al,2002). Those same capabilities can be used in the provider creation layer as well. However, service bus

mediation flow capability is not designed for the complex decision logic, exception handling, and state management required by the interaction scenarios you find in provider creation. On the other hand, other business process oriented products focus on complex flow capabilities intended for dealing with compositional and exception logic, an ability to hold, and represent in-process state for longer running processes. To summarize, for a successful SOA, you must architect layers with clean separation of concerns, then implement the layers retaining separation of concerns as much as possible. You can use any technology to implement a layer, whether the technology targets that layer or not, as long as compromises are understood and any impact on separation of concerns is minimized. Good governance during the model and assemble lifecycle phases goes a long way towards ensuring appropriate separation of concerns (Papazoglou and Heuvel, 2007).

RESULTS

THE NEW MODEL OF ESB ARCHITECTURE FOR ELECTRONIC COMMERCE SYSTEMS/PLATFORMS

In such a complex architecture, the ESB represents the piece of software that lives between the business applications and enables communication among them. Ideally, the ESB should be able to replace all direct contact with the applications on the bus, so that all communication takes place via the ESB. To achieve this objective, the ESB must encapsulate the functionality offered by its component applications in a meaningful way. This typically occurs through the use of an enterprise message model. The message model defines a standard set of messages that the ESB will both transmit and receive. When the ESB receives a message, it routes the message to the appropriate application. Often, because that application evolved without the same message-model, the ESB will have to transform the message into a format that the application can interpret. A software “adapter” fulfills the task of effecting these transformations (analogously to a physical adapter).This is illustrated in the figure 2 (a) below: ESB level Architecture based on Electronic-Commerce model implementation stage:

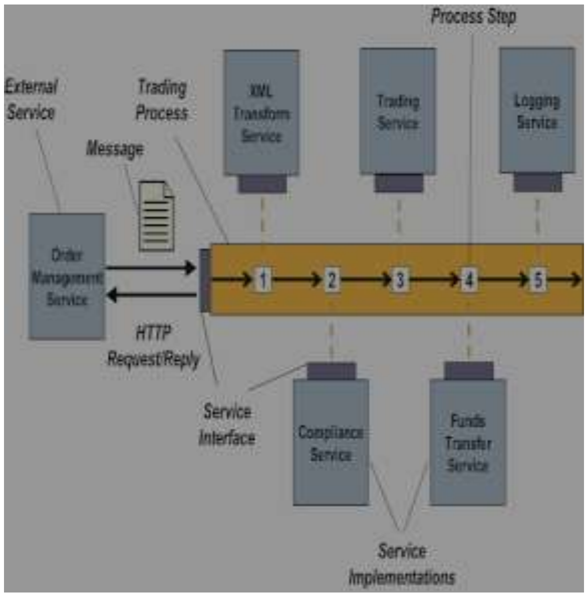

Figure 2(a) The ESB architecture based Electronic Commerce model

Characteristics of ESB architecture based Electronic-Commerce model.

Category	Functions
Invocation	support for synchronous and asynchronous transport protocols, service mapping (locating and binding services and processes).
Routing	addressability, static/deterministic routing protocols, content-based routing, rules-based routing, policy-based routing.
Mediation	adapters, protocol transformation, service mapping.
Messaging	message-processing, message transformation and message enhancement and remote procedure calls.
Process choreography	implementation of complex business processes and service protocols.
Service orchestration	coordination of multiple implementation services exposed as a single, aggregate service entity
Complex event processing	event-interpretation, correlation, pattern-matching
Other quality of service	security (encryption and signing), reliable delivery, transaction management.
Management	monitoring, audit, logging, metering, admin console, BAM

	(BAM is not a management capability in other words the ESB doesn't react to a specific threshold. It is a business service capability surfaced to end users.)

Table 1 (a) important features and characteristics of the implementation model of ESB

KEY BENEFITS

- Faster and cheaper accommodation and communication of existing systems and applications.
- Increased flexibility; easier to change as requirements change.

STANDARDS-BASED

- Scales from point-solutions to enterprise-wide deployment (distributed bus).
- Predefined ready-for-use service types.
- More configuration rather than integration coding.
- No central rules-engine, no central broker.
- Incremental patching with zero down-time; enterprise becomes "refactorable".

KEY DISADVANTAGE:

Increase overhead especially for those already compatible services and applications

DISCUSSION

Implementing an SOA requires a process of transformation and re-engineering. Each organization that deploys a SOA will go through this process in its own way based on that organization's unique requirements. Many organizations will begin to enable SOA by starting with a broker model. This will allow these organizations to initially focus on deployment instead of architecture, or application re-design. The SOA process encourages the re-use of existing security functionality as appropriate for local application deployment. Not all applications will be immediately deployable in a SOA model. Legacy functionality may need to be encapsulated within business logic instead of being exposed with a normalized interface. The model of the ESB will help ease this process. Application brokers implementing ESB functionality will provide a valuable piece of a SOA ecosystem; they will enable the service-identification of proprietary systems so that these systems can operate within an SOA. Implementing security across an SOA requires that one focuses on the conceptual security model across infrastructure components. Breaking down security functionality into security tasks allows us to build security services that can outlast application level implementations of security. A key part of this process is the establishment of a timeline to take architecture from a proprietary approach (fully application managed security) to a full, interoperable ESB based SOA. This timeline will often include the migration away from a proprietary application managed approach to security, through a reverse proxy pattern for HTTP (HyperText Transfer Protocol)-accessible resources, to a gateway pattern, for HTTP and non-HTTP resources, to a gateway as a full ESB. The ESB is designed to work within an existing environment. While it will allow for the consolidation of common services and functionality, it will not require the replacement/redeployment of a new infrastructure. These messaging services function well and do not need to be replaced just because a new architectural style has been introduced, or because new technology is available. Customers who already deploy these services should assess whether the existing topology already supplies sufficient security for the business while also assessing how to leverage the ESB to extend the reach of their service assets. An ESB provides the decoupling of service request and service invocation to support an SOA. It also includes the business events normally bound to the service invocations that can be logically associated with the message functionality, such as routing, message transformation and the underlying transport protocol conversion. The ESB is therefore a key enabler of a security-as-a-service model, in turn enabling a service-oriented approach to security infrastructure.The discussion here, as throughout the earlier article series, is about architecture, architectural layers, architectural roles, and architectural principles, such as separation of concerns. Architecture alone, however, cannot provide business value; the architecture must be implemented using appropriate technologies or products. Next, discussion focused on the pragmatics of architecting and implementing an SOA, the integration layer in particular. Separation of concerns is critical in SOA, and you need to define the necessary architectural layers to achieve a clean separation of concerns. That is the reason for the integration layer itself and for the mediation/service exposure and provider creation of sub-layers in the integration layer. A common cause of failed SOA implementations is failure to define a layered architecture that promotes separation of concerns, leading to an implementation that is inflexible and difficult to maintain. For example, some enterprises fail to appreciate the difference between an integration layer and a service bus, thus mixing service exposure and provider creation, in effect inappropriately mixing the responsibilities of business and IT. This does not mean, however, that an idealized layered architecture must be implemented with no compromises. Indeed, compromises almost always have to be made for reasons of performance, cost, technology limitations, and so on. For example, it is possible to use clean logical layering rather than physical layering to improve performance. The question is not whether to compromise, but what and where are the fewest possible compromises that retain the architectural intent and thus maximum separation of services. Obviously, to implement the architecture, you must choose an implementation technology for each of the architectural layers. In SOA (Service Oriented Architecture), there are often products that target a particular layer. Such a product will be very good for development, maintenance, and run time aspects of that layer, but not ideal for other layers. The technology choice for a layer might, however, be based on criteria in addition to the targeted architectural layers; the criteria can include skills, existing product inventory, development cost (tooling significantly impacts such costs), qualities of service, total cost of ownership, or a combination of these. It is perfectly valid to leverage a technology for a layer it does not target if other criteria are more important. The important consideration is not so much matching products to layers, but that the separation of concerns is preserved as much as possible by ensuring the separate architectural layers defined are still easily identified in the final solution.

CONCLUSIONS/RECOMMENDATIONS

I have presented two common patterns used in distributed electronic systems. The ESB (Enterprise Service Bus) has been used mostly for web services but it can be used for any distributed system. Enough details are made for an application designer or integrator to make good use of the functionality of these patterns. Even more precision comes from using "service specification" to refer to a formal representation of the external semantic, syntactic, and operational characteristics of a governed service and "service realization" to refer to a physical implementation of a service specification. I also showed that the term "service bus" in reality is logically a representation of (and physically a container for) a collection of mediations that perform semantically transparent "service exposure" that is, exposing providers according to service specifications. Service registry and repository provides the needed governance of (and can be considered a container for) service specifications. In addition to the above, service bus is not a complete integration layer, at least not an Enterprise Application Interface (EAI)-style integration layer that performs both semantic and non-semantic actions. I showed that mediation is a form of integration, and that the service bus provides a non-semantic "service exposure" sub-layer that is part of an integration layer in SOA that includes a "provider creation" sub-layer responsible for providing large-grained providers from one or more small-grained providers. Showing to be more precise in statements about "logic" in an integration layer in SOA and that business logic and integration logic are terms that can cause confusion, and identified business-owned semantic logic, business-owned non-semantic logic, and IT-owned non-semantic logic as more precise terms. I further showed the service exposure layer can contain only non-semantic logic, but that includes business-owned non-semantic logic, thus allowing business logic in the service bus (ESB). I also showed that, for a successful SOA, you should first architect the required layers that guarantee separation of concerns and then choose an implementation technology for the each of those layers, comprising only as much as necessary, thus retaining separation of concerns. A product targeting one architectural layer can be used for other layers as well, assuming preservation of separation of concerns. An ESB provides the decoupling of service request and service invocation to support an SOA (Service Oriented Architecture). It also includes the business events normally bound to the service invocations that can be logically associated with the message functionality, such as routing, message transformation and the underlying transport protocol conversion. The ESB (Enterprise Service Bus) is therefore a key enabler of a security-as-a-service model, in turn enabling a service-oriented approach to security infrastructure. The Enterprise Service Bus is type of multi-layer business model that seeks to separate the functions of the individual layers and processes involved in the request and response sessions within electronic commerce systems and applications as side the high security level and encapsulation interface it provides to various distributed service applications.

REFERENCES

[1]. Angelo Corsaro, "Quality of service in Publish/Subscribe middleware", http://www.omgwiki.org/dds/sites/default/files/Quality_of_Service_in_Publish-Subscribe.pdf

[2]. BEA Aqualogic Service Bus, http://en.wikipedia.org/wiki/AquaLogic(retrievedJune 27, 2011)

[3]. Braga, A., Rubira, C., and Dahab, R. 1998. Topic: A pattern language for cryptographic object-oriented software. Chapter 16 in Pattern Languages of ProgramDesign 4 (N. Harrison, [B] Foote, and H. Rohnert, Eds.). Also in Procs. of PLoP'98, DOI= http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/

[4]. David A. Chappell, Enterprise Service Bus, O'Reilly, 2004

[5]. D.F. Ferguson, D. Pilarinos, and J. Shewchuck, "The Internet Service Bus", The Architecture Journal 13, http://www.architecturejournal.net

[6]. D.F. Ferguson, D. Pilarinos, and J. Shewchuck, "The Internet Service Bus", The Architecture Journal 13, http://www.architecturejournal.net.

[7]. E.B.Fernandez, Sergio Mujica, and Francisca Valenzuela, "Two security patterns: Least Privilege and Secure Logger/Auditor.", Procs.of Asian PLoP 2011.

[8]. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal., Patternoriented software architecture, Wiley 1996.

[9]. G. Hoppe and B. Woolf, Enterprise integration patterns: Designing, building, and deploying message solutions, Addison-Wesley 2004.

[10]. J.P. Garcia-Gonzalez, Veronica Gacitua, and C. Pahl, "Service registry : a key piece for enhancing reuse in SOA service oriented architecture", The ArchitectureJournal;21,Microsoft, 2010. 29-36.

[11]. M. Fowler, Analysis patterns -- Reusable object models, Addison- Wesley, 1997.

[12]. M. Kircher and P. Jain, Pattern-oriented software architecture, vol. 3: Patterns for resource management, J. Wiley & Sons, 2004.

[13]. Soumen Chatterjee, "Messaging patterns in Service-Oriented Architectures"http://msdn.microsoft.com/en-us/library/aa480027.aspx.

[14]. SOA Patterns with BizTalk Server 2009, http://www.packtpub.com/soa-patternswith-biztalk-server-2009/book (retrieved on July 13, 2011).

[15]. Thomas Erl, SOA Design Patterns, Prentice Hall PTR; 1st edition, 2009.