Methods and tools for data and software integration - Enterprise Service Bus

Roman Hauptvogl
Department of Information Technologies
University of Economics, Prague
Czech Republic
hauptvogl@gmail.com

Abstract
Enterprise Service Bus (ESB) is one of the integration patterns and it is closely related to Service-oriented architecture (SOA) and to business integration. It is important to know the functionalities and the possibilities of ESB when we are thinking of its usage in the organization. In the following paper there is described the role of ESB in SOA. Architecture and common integration patterns which are closely related to ESB and its usage in enterprise integration are introduced. There are extended architectural principles for service design and there are shown the possibilities of the messaging framework which can provide additional features to ESB.

Keywords: Service-oriented architecture, SOA, Enterprise Service Bus, Integration, AquaLogic Service Bus, ESB best practices, Messaging Framework, Service Registry, Service Repository, SCA

1. Introduction

In many companies there are trends to moving to SOA. They invest a lot of financial and also non-financial resources to achieve benefits from SOA as soon as possible. Organizations are rethinking their environments and redesigning their architectures. They are creating coarse-grained, loosely coupled services and promoting service reuse, that are the basic principles of SOA [ERL, 2005].

In the last time a lot of books and articles appeared, which discuss various aspects of SOA. There can be found a lot of theoretical information, which can be used by the organizations for SOA implementation. One of the problems with successful SOA implementation is, that there is a need to learn not a single new technology but a whole series of different technologies. For example XML is technology that is used through the whole SOA, but with XML also come a lot of other supporting technologies like XML Schema, XQuery, XPath, XSLT and many more.

As described in [BEA, 2007] there is a huge potential of SOA in business integration. Because market today demands changes so rapidly, IT organizations need an intelligent infrastructure to gain service reuse and delivers reliable integration across the various technologies, protocols, and applications typical in today’s business IT environment.

In the organizations there are many different systems, applications and components which should be integrated or in other words, need to communicate together. There are already some open standards, which help us to integrate various systems communicating through various protocols. Typically, Web Services can be taken into account, which can exposed useful functionality through interfaces and build so reusable components also called services.
1.1 Goals of the paper

In this paper there is introduced an usage of the Enterprise Service Bus (ESB) in the enterprise integration, there are extended architectural principles for service design in ESB and there are shown the possibilities of the messaging framework, which can provide additional features to ESB. As a background for this paper I have used my theoretical and practical experiences in the establishment of SOA based integration environment in the former telecommunication and bank companies in Czech Republic. On these projects there have been used for integration mainly BEA (now after fusion ORACLE) based products like AquaLogic Service Bus (ALSB), AquaLogic Enterprise Repository (ALER), WebLogic Server (WLS), WebLogic Integration (WLI).

The main goal of the paper is to find out the possibilities of the usage of the ESB in the real environment. For the practical verification there have been used BEA's product ALSB. For this purpose there is need to provide partial goals:

- Definition of the basic principles of the ESB.
- Definition of the service architecture in ALSB.
- Definition of the messaging framework in ESB.
- Verification of the application of the well known VETO design pattern in ESB.
- Definition and verification of the usage of the new type of development process in ALSB.

1.2 Starting points and work hypothesis

There are many different definitions of the service. In [DAVIES, 2007] it is defined as “A facility providing the client with some benefit, such as food or electricity, or some capability, such as communication or access to information. Services act on behalf of the client, shielding the client from the implementation details.”. There were also defined basic principles of the service:

- Implementation is not part of the service.
- Everything that requires client knowledge is part of the service.
- Services are governed by contracts. Contracts define what the service does and how and when the service will be delivered.

As services proliferate in the SOA environment, the traditional approach have been to connect each service end point together, creating a sprawling web of point-to-point connections. While this initially seems to be the best way to address the connectivity, this web sprawls rapidly and becomes costly. Enterprise Service Bus (ESB) is specifically designed to eliminate such costly point-to-point service integrations, which limit IT’s ability to efficiently and quickly deploy services that support their SOA initiatives. Great example from the real world that describe this situation was presented in [JURIC, 2007] “We also cannot build connecting roads to other houses in any random way we like. This would result in one-to-one communications between houses that would be unmanageable; it would be almost impossible to oversee their construction and navigate between them. Rather, we again have to stick to a global design where it has been chosen how the roads will be positioned and interconnected.”
It is important to know that SOA is not only about redesigning and reimplementing all the systems and applications within the organization, but keep in mind that one of the basic principles of SOA is reuse. This reuse is mainly about creating components, designed in specific manner, that can be used several times. But this reuse can be also conceived as: “Reuse as much existing applications and components as possible.”. This is probably the hardest to understand, that there is need to decide which systems, applications or components in the company are ready to redesign and re-Implement, because it is nearly impossible to accomplish SOA like a big boom. It is iterative process, which needs to mature in the organization.

I have formulated several working hypothesis, which should be used to easier adoption of the ESB on the way to move to SOA. These hypothesis are researched in this paper:

- The architecture of the services in the ESB is not well defined and strict it can be exposed and need to be personalized for the company purposes.
- There are possibilities of the general messaging framework to manage business processes in ESB without usage of specialized Business Process Management (BPM) tools.
- Nowadays there are still some limitation and missing standards in providing design issues for services and there are also limitation in designing services in ESB.

1.3 Methods to achieve goals

The structure of the paper follow the basic principles of the academical work heading to practical application of the paper goals. In this first part of the paper there is introduced the theme and basic principles of the problematic. In the following three parts there are researched previously formulated hypothesis and in the last part there is conclusion of this paper.

There were analyzed available resources dealing with methods and tools for data and software integration and especially with ESB. There was analyzed the architecture of the services in ESB and the trends in the service design. With the use of the synthesis method there was designed an extension to the architecture of the services in ESB and there were created and formulated possibilities of the messaging framework which can extend basic functionalities of the ESB and this way provide some BPM features. In the next part of the paper there were analyzed the best practices for service design in ESB and verified its practical usage.

All the results of the paper were verified in the real environment and the customers included them to their process of the ESB-SOA adoption.

1.4 Basic principles

As stated in [JURIC, 2007] an ESB address requirements such as integrations between services and other middleware technologies and products, higher level of dependency, robustness, and security, management, and control of services and their communication. To this requirements it adds flexibility to communication, simplifies the integration and reuse of services.
The ESB is not a product used for a single project or for several types of projects. It should be part of the company's methodology to move to SOA and designed as a backbone of all the applications in the organization. This means that there is need to decide whether your organization is ready to deploy it in your IT environment or not. But this is not the only one decision of the IT in the organization. There is need to have a whole department, with sufficient number of specialized stuff, which will be solving problems of integration. Obviously, this will depend on the size of the company and the number of domains. The main role of this department is to prepare the plan to move to SOA integration. This is not an easy task at all. There are a lot of restrictions that needs to be keep in mind. One of them is that this department is embedded in organization which is solving every day problems and move forward in fulfilling designed goals. There is no place to stop this effort and start to solve integration. That is why there is need to have support from the management of the company and prepare the plan to move to SOA integration in wise steps and present the benefits of this integration. This plan need to be prepare carefully with a set of documents, which contains patterns, best practices, coding conventions and so on. These documents need to be prepared for different departments, which also need to see the sense in this strategic step. It is because the misunderstanding is one of the biggest thread of moving to SOA. ESB can play a big role in this movement, but to embedded it in the SOA environment as it was mentioned earlier is not an easy task. There is need to know the role of ESB and also to presents its benefits.

In general when communicating with services, there is on one side a client which is requesting the information or in another words it can be called service consumer. On the other side there is a service provider which provides desirable information. Both client consumer and service provider can reside in different environments, use different standards and also different communications protocols like HTTP(S), JMS, FTP, SMTP, RMI. There is need to solve the integration in the unified and standard based manner and that is the role of the Enterprise Service Bus (ESB).

ESB is usually embedded in existing integration environment so there is need at first to prepare a pilot project. In this pilot project there should be proved the abilities of the ESB and also the possibilities of its implementation. It is important to start to use ESB as soon as possible to see its benefits and to customize it for the company's needs.

There are a lot of different implementation of ESB. Although we can feel huge influence of the open source technologies in IT nowadays, bigger companies still prefer to choose one of the commercial solutions. This is mainly because with the commercial version comes also a support from the organization providing the solution which fill the gap of the unfamiliarity with the new product, what is desirable especially in early phases of the project. All the main vendors specializing on the enterprise applications, like BEA, IBM, TIBCO, Microsoft and many more, provide their own integration strategies and also provide platforms for realization of SOA based integration. ESB is one of the main product in SOA stack so it is also provided by all of them. In the following paragraph there is introduced Aqualogic Service Bus (ALSB) which is the product of the BEA Systems now Oracle.
1.5 AquaLogic Service Bus (ALSB)

AquaLogic Service Bus (ALSB) is a configuration-based, policy-driven ESB. It is targeted for service-oriented integration, managing Web Services (WS), and providing traditional message brokering across heterogeneous IT environments. It combines intelligent message brokering with service monitoring and administration in a unified software product.

From the development perspective ALSB does not follow the standard development life cycle, which among others usually consist of implementation, compile and deployment phase. The development process in ALSB is different in the way that it is configuration driven. It means that there is no need to work with source code, compile it and afterwards deploy your application to the application server, which has as usual its advantages and also disadvantages. First of all, the development is faster so IT can react more flexible to business needs. Changes are made within an active session and are propagated to the server as soon as the session is confirmed. There is no need to learn a new programming language. There is no need to spent time with the development, to learn the source code, to compile it and finally deploy resulting application to the application server. It is mainly about the configuration, which can help us to visualize the process flow. This process flow is not the business process, but the composition of the activities like validation, transformation, routing and so on. The responsibilities of these activities will be described later in this paper.

One of the reasons of this difference is because for the ESB should be also responsible the operation department of the company. This department should monitor this platform in run-time and should be responsible for the service contract, defining SLA (Service Level Agreement), error handling of the services and so on. And also visualize process flow can help in communication because it is another step to the effective business integration, the need of mutual communication between IT and business people. Because from the business perspective, SOA-based Business integration starts with the defined business objective and its associated processes. Business analysts use various modelling tools to sketch up or visualize the process flow. This way is for them much more easier to adapt to ALSB.

2. Architecture

The main purpose of the ESB is to provide infrastructure that connects disparate applications and other resources and provides flexible reuse of business components. ALSB consists of several resources which can be configured to create desirable integration. There are two fundamental resources in ALSB. First one is so called “proxy service”, which provides the interface to clients requesting service (some useful information). The proxy services also define what happens to the message when it enters the ALSB. Second one is “business service” which represent the enterprise service that provides some useful information. Both proxy and business services allow decoupling between clients and the back-end services. The architecture is shown on the Figure 1.
Methods and tools for data and software integration - Enterprise Service Bus

Figure 1: ALSB architecture

From the figure can be seen that there is need to connect to various types of different services. These services can be implemented using various technologies and can be placed in different platform environments. This is the main role of the ESB. For this purpose there are used various types of adapters, which provide connectivity to various services in uniform manner. These adapters usually consists of two parts: design-time and run-time part. Design-time part is used in the process of designing the service and it consists of the set of forms. It is usually configuration driven part. Run-time part should deals with the run-time questions like marshalling/unmarshalling, mapping data to the transport streams and so on. It is also important that these adapters should be part of the ESB as regards of look and feel and also usage.

Usually ESB products come with a set of, so called out of the box, adapters. Of course services should be implemented using proprietary technologies, so it is wise to provide also the mechanisms to create your own adapters. For this purpose there is need to provide the set of application programming interfaces (APIs). For example in ALSB it is used Transport SDK to build adapters that can be inserted into ALSB. In fact, transports that comes with ALSB out of the box are build using the same Transport SDK. One of the primary use cases for using custom adapters is for supporting a specialized transport a company already employs for communication among its internal application. This way higher interoperability is established.

One of the disadvantages of the proxy services is for example the situation, when there are several clients requesting the same service, but using different transport protocol. For this purpose there is need to design for each client its own proxy service, which is configured to use desirable transport protocol. This way there is problem with reuse of the logic implemented in the proxy service because there is need to implement the functionality again for every proxy service. With this
functionality it is meant for example validation, enrichment, transformation of the message and so on. Possible solution is to design the proxy which contains the functionality and for each client to create a routing proxy which will just receive the message from the client and then send it to the proxy service with the implemented functionality. In this solution there is need to pay attention to propagate properly meta data of the message (for example transport headers) and the possible error handling information. The solution is shown on the Figure 2.

Figure 2: Local proxy

With this solution can be a problem with the overhead, because there are spend resources needed for processing multiple calls. To solve this problem, there is possibility to use a so called local transport for the local proxy transport (the one which implements the logic), this way there is no need of marshalling/unmarshalling of the messages and the call to the proxy is done like normal call of the procedure in general programming language. Proxy with the local transport is not exposed to the clients so it can not be called directly outside of the ESB. This way some reusable modules can be also created. With this solution there is need to take care in designing of the error handling, because propagation of error handling information from one proxy to another is not trivial task. One way how to deal with this problem is to design the common error response message. Common message formats will be described in later sections.

2.1 Communication

The communications within ALSB is handled in several layers. When the message arrives to the service bus inbound transport layer is responsible for handling of this data. This data are raw bytes in form of input/output streams and can be bind to the different transport protocols. The inbound transport layer is simply for getting messages into ALSB and sending responses back. In the inbound binding layer all the data from the message are taken out and bind to the internal variables of the ALSB. The purpose of these variables is the message processing. They are in the XML form and represent message payload, message headers, attachment and transport specific parameters. Afterwards the processing of the message can be done in the proxy service. For the performance issues the attachments of the
message are provided to ESB only as a reference and there is no possibility to change the content of the attachment, if it is needed there must be developed a service which will be responsible for the management of the attachments. All the changes are made to the context variables with the use of common XML technologies like XQuery, XSLT, XPath. The processing of the message usually finished by routing message to the business service, which represent service provider. Sending of the message to the service provider is done in similar manner but in reverse order. At first the message is composed from the context variables, which is done in outbound binding layer. And then in outbound transport layer is message bind to the transport protocol in the form of input/output streams. The communication can be seen on the Figure 3. There is shown request and response path of the message. This way there is no need to take care of low level features, like binding to the transport protocol.

Figure 3: Communication within ALSB

ESB is a backbone of your company, so it should be reliable. Most of the messages within the application go through it, so it is good place where to place the monitoring of your systems. ALSB provides the capability to monitor and collect run-time information required for system operations and aggregates run-time statistics. Features that can be monitored are for example: min response time, max response time, overall average execution time, total number of messages, messages with errors, number of validation errors and so on. You can also establish service level agreements (SLAs) for the performance of your system, configuring rules that trigger automated alerts. If this alert arise you can be notified for example via mail. SLA guarantee a certain level of performance and also define the system response if those guarantees are not met.

2.2 Quality of Services

Quality of services should be the base of all messaging platforms and it is the basic need for any integration solution. Modern ESBs products have infrastructure based on the open standards mostly on XML technologies. Of course company can build its own product based on proprietary technologies but this way there is problem
with the interoperability and further reuse and extensions. It is because ESBs are positioned to the different environments and connects different applications. Nowadays when there are Web Services widely used and mostly set to the environment of internet, they also use internet's advantages and disadvantages. One of the most discussed problems are distributed transactions not based on the transport protocol. For example HTTP/S protocol used in the environment of the internet does not provide transaction support and event does not support quality of services reliable communication called exactly once, which means that the message is transported exactly once from client to service provider.

If we are in SOA environment we should not leave the quality of services on the transport protocol, because there is need to use some standard base, platform independent solution. The support of such standards should be obvious of all the ESB products. Also with the introduction of Web Service there was problem with the quality of services, such as security, transactions, and others. This gap was filled by introducing additional specifications:

- **WS-Security** – is a communication protocol standard released by Oasis-Open organization, providing a means for applying security to Web Services (originally developed by IBM, Microsoft and VeriSign). Specification can be found in [OASIS, 2004].
- **Transaction specification** – this specification was created in cooperation of firms BEA, IBM and Microsoft and defines the mechanisms of transactional cooperation between services. It consists of:
  - **WS-Coordination** – defines the coordination framework for the Web Services to coordinate distributed actions and also provides tools for data synchronization
  - **WS-AtomicTransaction** – specifies short duration ACID transactions
  - **WS-BusinessActivity** – specifies longer running business transactions
- **WS-Reliable Messaging** – provides reliable communication over various transport protocols
- **WS-Addressing** – specifies message coordination and routing
- **WS-Inspection** – provides support for dynamic introspection of web services descriptions
- **WS-Policy** – specifies how policies are declared and exchanged between collaborating web services
- **WS-Eventing** – defines an event model for asynchronous notification
- and many more

Cooperation of such big firms such as IBM, BEA, Microsoft in defining additional standards for Web Services proves the interest and wide spread of this technology and the need to develope addition supporting standards to solve the open issues in this technology.

### 2.3 Environment

There have been a lot of articles written about the need to find out a way how business and IT professionals can together collaborate and work individually across the extended enterprise. ESB stand alone cannot fulfil this task, but it is not
separable part of the set of products, which can help to build the unified platform for SOA, the product ESB should be able to cooperate with various different products from this set, because it is positioned in the middle of this set.

BEA for example believes that its set of products called WorkSpace 360 will deliver the truly shared workspaces that empower the most important Service-Oriented Architecture (SOA) leadership roles across the enterprise, like the business analyst, enterprise architect, service/application developer and the IT operations professional. Its role is to provide unified eclipse based environment for all of them, this can help in quick adoption of new products and provide easier collaboration. To have one unified environment there is need to stores all the meta data of all the enterprise assets. For this purpose there is used Enterprise Repository which is described in the following section.

2.3.1 Service Registry, Enterprise Repository

Usually organizations are not sure whether to use Service Registry, Enterprise Repository or none of them and also what is different between these two technologies. In the following section I will try to provide several hints for usage of Service Registry and Enterprise Repository.

Service Registry stores the information (meta data) about the various services which are used for the service searching. Service Registry is suggested only for large enterprises, where the number of services can be large and become mandatory requirement in the case of B2B solutions. It is because, there is need to make enterprise services publicly know and give the possibility for the consumer to look up the service registry for a desired services. Usually the ESB products have the support for the various Service Registries, what means that when there is a service deployed to ESB it can be also automatically publish to previously configured Service Registry and also changes made to the service interface in ESB are also automatically publish to Service Registry.

Enterprise Repository provides the tools to manage and govern the meta data for any type of software asset, from business processes and services to patterns, frameworks, applications, components, and data services. In general we can say that Service Registry is used in run time to look up services and Enterprise Repository is mostly used in design time to design services and also composite services to gain as much reuse as possible. It means that Registry and Repository can be used inseparably to provide a core for the SOA governance to keep SOA on track. The solution can help to provide visibility and traceability of an organization’s entire service and asset portfolio, where the registry and repository provide centralized management of meta data associated with composite application components, as well as applied governance policies.

2.3.2 Service Component Architecture

Service Component Architecture (SCA) is a specification for building and packaging applications and allows developers to focus on writing business logic. This specification was created with cooperation of organizations BEA, IBM, Oracle, SAP, IONA, Siebel, and Sybase. The goal of SCA is to make building applications easier and it has been designed for SOA.

SCA uses the concept of assembly to solve key problems presented by SOA development including:
Separation of business logic from underlying infrastructure, qualities of service, and transport

Linking programming in the small with programming in the large

Providing a unified way to move to and from architectural design, coding, and operational deployment in a bottom-up or top-down approach.

SCA is again another step to provider easier cooperation between business and IT professionals in the movement to SOA.

2.4 Summary

With the use of routing proxies and proxy services with local transport protocol can be architecture of the services in ESB easily exposed and this way can be also created reusable components. But these components are not the business ones, which should be implemented as services, but for example some reusable custom validations, transformation and so on. Routing proxy can server as an unified gate of the requests from one type of the consumer and there can be used the content base routing, which is one of the basic features of ESB, to route the request to the desirable proxy which will manage the request. This way we help to minimize the interdependencies between service consumer and service provider.

The manipulation with the data within the component is in the unified manner using XML technologies. With the use of different technologies like Service Repository, Service Component Architecture can be extended design part of the services and to gain better reusability of the services within the organization.

3. Custom Messaging Framework

An increasing number of services naturally results in an increasingly complex environment. There is need to have this complex environment under control. Therefore one of the basic needs is to monitor the messages transferred through the system. When there is used pattern in your organization that all the valuable services are exposed through the ESB then the most natural point where to monitor services is exactly on the ESB, because all the queries from the consumers to service provider goes through it. It is true that every ESB product have it's own monitoring solution, used in operational (run-time) phase of the service life cycle. It is used to observe the run-time behavior of the service. Monitoring can provide quantitative data about service execution times, number of requests, number of error responses, and many more. The outputs from the monitoring can be further used for analysis and optimization of service behaviour and performance.

But with the use of meta data within the transferred messages the monitoring framework can be extended much more. For this purpose there is need to define a unified set of meta data, which will be used in each service. There are several ways how to store meta data in run-time execution of the service. For example if we are using web services, there is used SOAP protocol for transport, SOAP message headers are specially designed for this purpose. But when there is used for example jdbc or some custom transport there could not be implemented such support. For unified work with services in ESB, where can be service customers and service providers settled in the different environment and for communication use different technologies, are all the messages sent through ESB serialized to
XML messages and on the output deserialized to desired transport format. For this serialization and deserialization is responsible binding layer mention in chapter “Communication”.

For organization where there are used several different and event some custom technologies for communication there is an interesting vision, which should be taken into consideration, to design and create the common format of all the messages sent through the ESB with a set of meta data. It is not trivial task and there is need to have support of all the system connected to the ESB. With common format it is meant the common format of the meta data of the message, of course the payload of the message will be different and dependent on the functionality of the service. But if you will be successful in this task, you can gain some additional benefits from ESB. This meta data can be used for example on custom logging, monitoring, profiling, error handling and so on. In the following paragraphs there are described the extensions to messaging framework for custom business process monitoring, custom error handling.

3.1 Custom business process monitoring

The purpose of the ESB is not to take care of the whole business process but only to provide connection between one client and one service. But business process can contain several services and also clients. If there are used meta data within the message that identify the process this way you can monitor and control it. An example of this meta data type can be for example business process identification, which identify the business process. This meta data can be used for monitoring whole business process without support of some BPM tool.

Another important data for message exchange is the message correlation. Services in ESB are designed as stateless and usually when there are build complex business processes, there are usually called more then one services. For this purpose it is clever to store identification of the correlation within the message. Imagine example that client uses JMS for invoking the service on ESB. If this was simple call ESB products have usually some tools to handle the protocol headers as for example JMS Correlation ID, but if this process will be complex and asynchronous, there is no way where to store the JMS Correlation ID which will be used on the client side for message correlation. Another example is when there is for example JMS Message ID used as a pattern for correlation. You have to keep in mind that JMS Message ID has different formats in different environment, so even if you use some message bridge between different environment the JMS Message ID will be different and that is why you cannot used it for correlation.

3.2 Error handling

With the usage of the meta data can company event build it’s own complex error handling solution. For example imagine that client sends asynchronous message to the service that is currently not available or even there was some change in the service interface. In general you can usually setup in ESB environment number of the resends for the service, the interval between the resends and the error destination if even resends where not successful. The framework with the use of meta data can log the message and save it for further resend. This meta data can for example identified the service provider and can be used for the identification of
the information needed for the message resent. There can be used reporting tool that will notify administrator about the error. Then the administrator will have the possibility to resent the message after the cause of the problem is already solved. Even there is sense to provide to the administrator tool for modification of such messages in the unified manner. Of course this process of resending can be also automatized when there is not need of the human intervention.

3.3 Summary

Because with the usage of the ESB we are able to connect different services and compose them in the business flows, with the monitoring framework extension it is possible to monitor these business flows. With the custom error handling solution provided by messaging framework it is possible to handle, to some distance, errors. The result is the possibility of to usage of the ESB product with the combination of the messaging framework to manage custom business processes. Of course for this purpose there are BPM products that can do these tasks naturally, in unified and standard based manner, they have even environment to visualised the business processes, but for several easier processes and on the beggning of the way to adopt SOA it is even possible to use ESB.

4. Best Practices and Design Patterns

As it was already mention earlier the main principle of the ESB is to have applications connected to the service bus in the manner of loosely coupled services that do not have direct knowledge of each other. It is very important, because even if there are changes in the service logic, service consumer do not need to know about them and even do not need to react on them. It is ESB's responsibility to ensure that the service providers and service consumers communicate together in loosely coupled manner. In another words consistent data need to be passed within the service bus. To ensure that service consumer and provider communicate in loosely coupled manner there is need to adhere service design principles. As it is stated in [FEUERLICHT, 2008] there are still some missing standards for the service design. There are in general some design principles which can be taken into consideration for example:

- maximization of cohesion – implement a single conceptual task
- minimization of coupling – minimize the interdependencies between services
- orthogonality – functionality of the service does not overlap

The problem is, that there are no widely accepted service design methodologies. In the previous chapters there was described an architecture and its extension available for the service design and the messaging framework which can help us to fulfil the purpose of loosely coupled communication. Because general service design methodology would cover whole new book in this chapter I would like to introduce some best practices for designing services in ESB.
4.1 VETO Pattern

There is a common integration pattern which is delegated to ESB and can ensure that consistent and valid data will be passed through ESB. This pattern is known as VETO pattern. It is acronym made from the first letters of the activities “Validate”, “Enrich”, “Transform” and “Operate”. As it was mention in previous chapters, all the context variables including message payload are in the form of XML. It means that to perform activities included in VETO we will be mostly working with XML technologies.

- The validation of the incoming message should be done as soon as the message arrives to the ESB. Its role is to check whether incoming message contains a well-formed XML document and conforms to a particular XML schema or WSDL document that describes the message. If we will perform this task later after some processing, we could find out that we are missing some element, which will lead to rejecting of the message and useless previous processing. Validation may be required to guarantee the reliability of an XML application. However validation can have significant effect on performance, so it is needed to take into account certain considerations:
  - messages exchanged within internal systems may not require validation
  - messages coming from untrusted external sources (included human composed messages) must be validated
  - messages coming from external source, once validated, may be exchanged without further validation

- The enrichment of the message involves adding of an additionally needed elements, from an external source, to the message to make it more meaningful to a target service or application. This action can be performed by calling another services and receives so additional informations.

- The transform step is responsible for converting of the message to the target format. It is important to transform message to the valid format in order to prevent target service from rejecting it, because it can have its own validations process.

- The operate step is used to send the message to the target service. This step can include situation where are messages sent to concrete destination, but can also include content base routing, which means that the message is sent to the destination according to its content this way there can be intelligent routing created.

As it was mentioned before the VETO pattern is the basic pattern for the ESB. This pattern is generally used but organization need to adapt it to its needs. If we decide to used this pattern then all the services deployed in the ESB environment should conform to it. But the main problem of this pattern is its reuse in implementation. In general if there is prepared one service, which will be used as a pattern for all other services, there must be paid a lot of attention in its design, because if you once use some pattern for all of the services, there is no easy way to change the pattern, because it is glued with the each proxy service and it is not configuration driven. There is no way to configure more than one service at a time.
But VETO pattern is only a skeleton for your services and there will be usually inserted additional actions within the actions of the VETO pattern like initialization, auditing, logging, error handling and so on. Initialization can be useful if you are using the same data within the service, which are taken out from the message using for example XPath technology. It is wise to prepare the data and copy them to variables on one place and then use only these variables. This is also case of the service meta data which are stored within the message and these variables can be used later on for example in logging action. This way you can prepare a component for logging and reuse it in another services.

### 4.2 Split-Join Flow Pattern

I have chosen the following pattern to demonstrate that there are situations when there is need to use also different products to solve some problems. All the services implemented on the ESB are stateless. There are different integration products and also technologies, like for example Business Process Management (BPM) and Enterprise Java Bean (EJB), which are responsible for implementation of the statefull processes. This process can sometimes execute for a longer period of time (event days), where it is particularly important to provide a transactional support. This type of services are not an ESB responsibility.

But imagine an example where there are large messages which require processing of different portions of the message, which can be split into smaller messages. This smaller messages are routed asynchronously to the target services according to the content where they are processed. But for creating the response message back to the client there is need to develop a new aggregate service, which will be responsible for collecting the responses from all the target services and sending message back to the service bus. Service bus will process the message and sends the response message to the client.

### 4.3 Development process

Development process in ALSB is different in the way that it is configuration driven, this mean that there is no need to work with the source code. It is important because services on ESB are accessible not only for developers, but also for analysts, architects, service designers and so on.

All the work in the ALSB is done within an active session. When the session is confirmed, the changes will take an effect. Of course if there are some conflicts or some invalid configurations within the session, session is not allowed to be confirmed. Each confirmed session can have a short description. The changes in the configurations can be seen in the form of the sessions logs, which are these descriptions of the sessions. But the problem is, that it is the only way how to see the changes. What I found not sufficient, especially when you are working in a team. This could be solved by defining the responsible person for the group of the services. What means that there will be person responsible for confirming his/her defined services. All the services could be developed on local machines but there is need to have one reference machine, where there is also need to confirm the services. On this machine, for example in Development or Test environment will be all the services actual and updated.
Each confirmed session can be discarded, which will be resulting in the change of the state of the ALSB to the state before the session was started. It is important to choose the description of the session so, that we will recognize the session and in case of problems could execute the rollback.

Also the point of view on version control is different, but you need to keep in mind that implementation is not part of the service so there is no need to have common version control system. With version control of our services can helps us some of the SOA governance systems like Enterprise Repository or Service Registry. They can bring us an effective SOA governance. It keeps SOA on the track to deliver maximum reuse, maximum agility, and maximum Return on Investments (ROI). It addresses common SOA challenges:

- Lack of visibility into planned and existing services
- Growing complexities in the enterprise, both business and IT
- Inability to manage the impact of change
- Limited, if any, insight into demonstrated savings, ROI and benefits

If there is no such SOA governance system it is at least wise to version control the service artifacts like WSDL, XSD, policy files and so on.

Like in the normal development process there are best practices which also can be applied for development process on ALSB. It is wise to set up the coding conventions, naming conventions, folder structure and so on. It can help us to get uniform environment. Another thing is to prepare the scripts for moving from development to test environment and from test to production environment. This scripts can help us to set up the proper endpoints of our services. For this purpose can be used so called WebLogic Scripting Tool (WLST). It is BEA tool for scripting the configuration of BEA domains. The secret of its power is its scripting language, Jython, an implementation of the popular Python language in Java.

What I found as the main disadvantages of this configuration driven development are:

- Developer is not able to see the source code of the service
- Problem of working in teams
- Problem with the support of version control system
- Integration with another development tools
- Not able to use the scripts for mass editing

In the latest version of the ALSB product, there was introduced new eclipse based IDE for service development which solved some of this problems. This way can be also separated the phase of development from the operational (run-time) phase.

### 4.4 Summary

Even thought the development process in ALSB is easier and faster in compare to normal programming environment, but it have also some limitation, for example in version control, mass editing and also in adoption of the design patterns. This means that patterns need to be used wise and need to be adapted to the company's needs. There is also golden rule for the ESB services, which should not be used to solved business functionalities, these functionalities should be placed in
the business services and ESB should only provide the access to them for the consumers.

5. Conclusion

In this paper I have tried to describe the role of ESB in SOA and its possibilities of the usage. As it was mentioned several times, the main purpose of the ESB is to provide infrastructure that connects disparate application and other resources and provides flexible reuse of business components. I have tried to introduce and to solve some common problems which can be taken into consideration when using ESB.

From my own experience I can say that implementation of the services on ALSB is very fast and effective, but there are still missing some widely accepted standards for the service design, service composition, granularity of the services and so on. In general are nowadays design principles formulated by each organization itself.

The deployment of the ALSB product to real environment is straight forward and can be extended to solve some addition requirements, like for example BPM requirements. I think that ESB is not designed for all the companies, but it is designed for those, which have quite a lot of different services spread through the different environments and these services are still evolving. ESB has special position in the SOA environment. Where I see the strength of the ESB is in its simplicity and rapid response to the business needs by quick configuration of routing rules based on changes in business rules.

5.1 Target analysis

The main goal of this paper was to provide basic principles and the usage of ESB in the enterprise integration, followed by extension of the architectural principles for service design in ESB and to show the possibilities of the messaging framework, which can provide additional features to ESB.

Goal was achieved by:

- There were shown the possibilities of the extension of the service architecture in the chapter 2.
- There were created basic principles of the messaging framework and its usage for managing of the business processes in the chapter 3
- There were described and shown the problems connected with the design principles of the services in the ESB in the chapter 4

5.2 Practical usage of the goals

Outputs from the paper are in use in the former telecommunication and bank companies in Czech Republic and help to easier adoption of the SOA.

5.3 Topics for the further research

From the paper it is evident that there are still missing and widely accepted standards for

- general service design methodologies
6. References