Application integration using Enterprise Service Bus products

Milan Rubeš
Katedra informačních technologií
Vysoká škola ekonomická v Praze
Milan.Rubes@seznam.cz

Abstract: Many companies with huge application portfolio are facing the problem of inflexible IT and high expenses spent on application development. They are planning to adopt service oriented architecture which in integration area usually leads to implementation of enterprise service bus or similar product. There are many pitfalls that can happen and not every way leads to successful implementation and does not reach the expectations. Author of the article focuses on integration based on ESB product as this approach becomes the most used one at present time. The author also draws upon his experience with application integration in several banks in Central and Eastern Europe. The article describes the ESB product implementation from the first thought to deployment to production environment.

Keywords: application integration, enterprise service bus (ESB), web service (WS), service oriented architecture (SOA), business process management (BPM), model driven architecture (MDA), IT governance

1. Recent ways of application integration

Once companies started to build their applications to support business needs integration with other companies’ or partners’ applications appeared. From logical point of view, application integration is usually done on integration layer regardless whether the applications are built as multi-tier applications or not. For the very beginning peer-to-peer seemed to solve the problem of application integration.

1.1 Peer-to-peer integration

Peer-to-peer integration is based on direct interconnection of applications. Service consumer has to understand all service providers’ interfaces and has to be able to use all service providers’ transfer protocol to obtain the required information. Following figure illustrates peer-to-peer integration in financial sector.
1.2 **Message oriented middleware**

Integration using message oriented middleware (MOM) brings the governance to application integration. The integration layer provides an infrastructure that removes any direct (point-to-point) connection between services and minimizes dependencies between application layers.

1.3 **Other ways of integration**

There are also other ways how to integrate applications focusing on particular communication techniques like CORBA or RPC or focusing completely different approach like data sharing, data replication and others that are not further discussed in this article.
2. Requirements for integration changes

As today’s requirements for application integration grow so quickly many architects put more stress to improving integration principles. New principles will not probably reduce the complexity of enterprise application portfolio but the integration complexity can become more efficient by focusing on integration governance.

2.1 Managing complexity

Computing environment in medium or big enterprise consists of heterogeneous running on different hardware platforms, operating systems, network interfaces, middleware, and communication protocols.

The strategic plan is to reduce the complexity but it is clear that number of different platforms from different vendors (both HW and SW) will still have to run within the enterprise IT environment.

To be able to manage the complexity flexible architecture standards and patterns have to be set up including technology and organizational aspects.

The architecture has to be able to adapt to varying environment, hide heterogeneity from applications, and transparently and dynamically switch between technologies.

2.2 Lowering expenses

As the complexity of the application integration grows, total number of interfaces grows nearly in geometric way. The corresponding costs (maintenance, personal, license, hardware and other) grows appropriately. This growth is in contradiction of enterprise requirement to reduce overall company’s expenditures.

2.3 Shortening application lifecycle

Even harder business competition force companies to bring new products to market much more quickly. Shortening product lifecycle also shortens lifecycle of the applications being used to support the product.

3. Service oriented architecture principles

All the above mentioned problems lead to form new principles of building applications and integrating them. These principles form Service oriented architecture (SOA) which can be defined as follows [BEA]:

Service-Oriented Architecture is an IT strategy that organizes the discrete functions contained in enterprise applications into interoperable, standards-based services that may be combined and reused quickly to meet business needs.

3.1 SOA Principles

ESB products are implemented to support or enable SOA. Common SOA principles that are applicable to almost every ESB product are as follows [WIKISOA]:

- Reuse, granularity, modularity, composability, componentization, and interoperability
- Compliance to standards (common or industry-specific)
• Services identification and categorization, provisioning and delivery, and monitoring and tracking

The following specific architectural principles for design and service definition focus on specific themes that influence the intrinsic behaviour of a system and the style of its design:

• Service Encapsulation
• Service loose coupling - Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other
• Service contract - Services adhere to a communications agreement, as defined collectively by one or more service description documents
• Service abstraction - Beyond what is described in the service contract, services hide logic from the outside world
• Service reusability - Logic is divided into services with the intention of promoting reuse
• Service contract - Collections of services can be coordinated and assembled to form composite services
• Service autonomy – Services have control over the logic they encapsulate
• Service optimization – All else equal, high-quality services are generally considered preferable to low-quality ones
• Service discoverability – Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms

4. ESB product based integration

4.1 ESB products overview

This chapter brings an ESB definition based on [NWC] and [WIKI]:

Enterprise service bus is an application infrastructure that joins disparate services and ensures availability of those services. It enables business services to be turned into business processes by business process management (BPM).

ESB does not implement a service-oriented architecture (SOA) but provides the fundamentals for successful implementation. ESB should be standards-based and flexible, supporting many transport protocols.

4.2 ESB product features

There is no clear definition which features should ESB product contain. Therefore vendors call their SOA infrastructure products “ESBs” even if their capabilities are limited only to communication tasks. ESB products always have the core bus and they typically include following features [GARTESB]:

• Message transformation to accommodate different message formats used by applications
• Content-based routing to send messages to appropriate service according to its content
Application integration using Enterprise Service Bus products

- Service registry to store run-time information as well metadata about the services
- Service orchestration to implement short-time message flows
- Security, including authentication and authorization
- Optional adapters to files, databases, legacy platforms and packaged applications

They often also have:
- Message validation (based on XSD definitions)
- Transaction management capabilities (support of XA protocol)
- Message logging and auditing for debugging and security reasons
- Protocol bridging to change the protocol used by one application to another one used by another application
- Load balancing and failover mechanisms

5. ESB product types

ESB products can be divided into several categories according to parameters we are interested in. As company businesses differ also different products can satisfy different business requirements. At the end of the chapter there is a short summary with recommendation which one to use.

5.1 Product origin criteria

Main division of ESB products may be done according the product origin – “SOA enabled MOM” and “New SOA based suite”.

5.1.1 SOA enabled MOM [S]

Many of the vendors built their ESB on proven old product, just adding some features like web-service support and renamed it to ESB or SOA suite. The following figure shows typical architecture of product of this kind. It consists of monolithic core which can be viewed as a black box.

Figure 3: SOA enabled MOM [CSESB]

Typical products of this category are IBM WebSphere Message Broker, Tibco BusinessWorks, Sonic SOA Suite.
5.1.2 New SOA based suite [N]

Other vendors built their product entirely from a scratch based on the best known SOA principles. Their solutions are modular and internally SOA based which means that particular features can be replaced by other products.

![Diagram of SOA based suite][1]

Figure 4: New SOA based suite [CSESBB]

Typical products of this category are BEA Systems SOA 360, JBoss Enterprise Middleware Suite, Oracle Fusion

<table>
<thead>
<tr>
<th>Product type</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOA enabled MOM</td>
<td>Based on proven solution</td>
<td>Limited set of features</td>
</tr>
<tr>
<td></td>
<td>Powerfull</td>
<td>Lack of supported standards</td>
</tr>
<tr>
<td></td>
<td>Good support of the product</td>
<td>inflexible</td>
</tr>
<tr>
<td>New SOA based suite</td>
<td>Standards based</td>
<td>Full of bugs</td>
</tr>
<tr>
<td></td>
<td>Built on best practise</td>
<td>Little or poor references</td>
</tr>
</tbody>
</table>

Table 1: Comparison of product origin

5.2 Product usage criteria

Another division may be done according to planned ESB usage – whether the company is willing to integrate in-house developed applications, with a possibility to make changes to them, or just integrate packaged applications. [GARTUSE]

5.2.1 Uniform design [U]

Most of the services in this scenario are locally designed and custom built, so the appropriate SOA infrastructure should bring good IT governance, obligatory architecture patterns, support for old transport protocols and messages (non-XML: COBOL copy book, binary etc.)

Typical products of this category are, BEA Systems SOA 360, IBM SOA Foundation, JBoss Enterprise Middleware Suite, Oracle Fusion.

5.2.2 Integration of packaged applications [I]

These products have to be able to support integration-centric scenarios which means integration of packaged (not custom made) applications. Main features for this scenario are message transformation, adapters to packaged applications and legacy platforms, protocol bridges and database adapters.
Typical products of this category are Apache/Iona Fuse and Celtix Enterprise, Cape Clear 7 Server, IBM WebSphere Message Broker, Sonic Software Sonic SOA Suite, Tibco BusinessWorks and ActiveMatrix.

<table>
<thead>
<tr>
<th>Product type</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform design</td>
<td>Useful for large SOA enabled packaged applications. Brings good IT governance.</td>
<td>Impossible to force pattern usage for external based applications or services.</td>
</tr>
<tr>
<td>Integration of packaged applications</td>
<td>Useful for packages from two or more vendors, many legacy applications or many externally based services.</td>
<td>Unsolved complexity of the integration.</td>
</tr>
</tbody>
</table>

Table 2: Comparison of product usage

5.3 Product selection

The choice of SOA infrastructure can be made tactically (project-by-project) or strategically (once, for all the projects within a large SOA). However, large companies will hardly ever be able to make all the projects in all their business units use the same SOA infrastructure technology for a variety of technical and organizational reasons. Most large companies will have some projects and business units that are primarily uniform and others that require substantial integration, so the most common, and most appropriate, solution in those companies will be to implement two or more SOA infrastructure technologies to optimize for each scenario. [GARTUSE]

Table 3 brings recommendation for ESB product selection based on future usage.

<table>
<thead>
<tr>
<th>Product type combination</th>
<th>Recommended usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>S&amp;U</td>
<td>Not recommended</td>
</tr>
<tr>
<td>S&gt;I</td>
<td>Unchanging applications environment Focus on performance</td>
</tr>
<tr>
<td>N&amp;U</td>
<td>SOA enabled big application Many home made services to be integrated</td>
</tr>
<tr>
<td>N&gt;I</td>
<td>Integration of standard applications (no in-house development)</td>
</tr>
</tbody>
</table>

Table 3: Product selection recommendation

6. ESB implementation process

ESB implementation process is not as easy as simple application introduction. It is complex activity composed of several steps. Based on experience with ESB product implementation, author of this article divided the implementation process into below described steps. These steps may be done only respecting others but parallel implementation for some of them is possible.
6.1 Identification of reusable services

In a pre-service oriented development environment reusable services might be performed by reusable code libraries or class libraries that are loaded or linked into new applications. In SOA-based applications, common functions such as these, as well as typical system functions such as security checks, transaction coordination, and auditing are instead implemented using services. Using services not only reduces the amount of deployed code, but it also reduces the management, maintenance, and support burden by centralizing the deployed code and managing access to it. [AWP01]

Typical examples of reusable services may be customer name lookup, zip code validation, account validation, or credit checking.

Identifying reusable services depends on company's business and current architecture. If more systems provide the same functionality rebuilding and integrating that functionality as reusable services is a good start.

6.1.1 Service granularity

Services may be implemented with different level of granularity. Some of them may be too general which means that the functionality covered by one interface is too wide. We are implementing ESB to support as much reusability as possible so the better way is to implement services covering the smallest piece of functionality to be able to build complex services by composing them from the granular ones.

This approach also allows fast implementation changes when business processes are changing to support new products or services. In that case new services are built by recomposing singular services with no additional effort.

6.2 Application service-enablement

Key focus of this step is to service-enable legacy functionality for direct consumption by non-legacy applications. The way of service-enablement depends on the environment which the application is running on and on application itself. There are several ways of service-enabling applications mentioned below.

6.2.1 Changes to applications

If the application is developed in-house it is possible to change and restructure the code to create independent blocks of code and after that to implement a wrapper or apply a general implemented wrapper.

6.2.2 Service enablement using 3rd party product

There are several products on the market which automatically service-enable applications on supported environments. These products are mainly focused on mainframe applications. Examples of those products are IBM's WebSphere Information Integrator Classic Federation, iWay Software's Adaptive Framework, GT Software's Ivory Data Access.
6.2.3 Service-enablement through application rehosting

Service-enablement can be joined with application rehosting to other platform. This scenario is quite complicated but in certain cases it may be the only way how to effectively service-enable legacy applications. There are several products that can be used to automatically rehost application from one system to another for example from mainframe system to UNIX or UNIX to Windows without changing the code of the application.

6.3 ESB product implementation

First of all, the core ESB product has to be implemented and/or customized. Most of the products are “out-of-the-box” ones so only configuration is needed. For the implementation it is worth to have a vendor’s expert on-site to quickly identify all possible problems.

In this step, implementation team will probably have to face many problems including bugs or unfinished features in ESB products as they are not mature enough. At least a small amount of development will be needed to enhance the basic features offered by the product.

6.4 BPM flows definition

In parallel with product implementation BPM flows could be defined. Flow definition is based on business requirements so precise definition of the requirements is the prerequisite for successful implementation.

Using SPSPR model developed at VŠE [VOR] BPM flows definition are "ICT-Service value" representation of the "Business process value" layer implemented by "ICS Process value". In other words, BPM flows represent particular company processes implemented by ICT services that are composed of ICT processes.

6.5 Security, management and monitoring settings

After the product is implemented, security, management and monitoring parameters have to be configured.

6.5.1 Security

Limited access to services and applications is most common way of securing ESB applications and services offered by ESB. Most of the products offer authentication, authorization, encryption and other security settings. Some of them (for example BEA AquaLogic) offers also possibility to manage access to services based on date/time, service invocation count and other fine-grained parameters which usage can lead to complex security settings.

6.5.2 Management

Most of the products offer web-based management console. It allows full or partial access to product settings. More important than having management console is to have a possibility to configure the system using configuration files. These files (often based on XML) allow administrator to perform automatic migration from one
environment to another using scripts. This way of migrating is safer than configuring parameters via web-based console.

6.5.3 Monitoring

To precede system failures and instability detailed application monitoring should be implemented. Using monitoring products (for example HP OpenView) can help to be proactive and to avoid system failures.

Monitoring of the ESB product is only part of the overall information system monitoring but ESB has the special position as most of the problems are immediately propagated to this layer.

Most of the products have their own monitoring included accessible by web-based management console and the major products (BEA AquaLogic, Oracle ESB, IBM Message Broker and others) are also integrated with 3rd party management and monitoring systems.

ESB products offer the possibility to setup many metrics to be monitored, for example infrastructure metrics (CPU, RAM, LAN, disks, ...), performance metrics (service invocation count, message size, ...), statistical metrics (service invoked by user, service called by service, connections, ...) and many others.

6.6 Testing

After successful implementation testing has to be done. Testing depth depends on importance of the application integration in particular company.

6.6.1 Technical testing

First phase of testing, in some projects part of the development, is technical testing of the solution. In this phase companies usually test connectivity to other applications, proper protocol conversions, service invocation and security settings.

6.6.2 Functional testing

Functional testing verifies that the implemented functionality matches the required one. Owners of the applications integrated by ESB should test the functionality for correctness and perfection. The ESB cannot be tested directly as individual product but it is tested throughout testing connected applications.

Correct functionality for ESB means that all the flows, message transformation, data mapping and conversions are implemented according to the specification.

6.7 Pilot

Pilot phase is in usual way used to test new version of application in production environment. New version is being tested by selected group of users only.

This approach is not applicable to test new integration solution as it is not possible to split integration processing according to selected users. Pilot phase for ESB may be realized by selecting limited set of applications (for example the less important ones) to be tested using newly implemented ESB.
6.8 Roll-out
If pilot phase is used in implementation life-cycle, roll-out is a simple step when all the remaining applications are configured to use the ESB.

7. Time plan
In mid and large companies with lot of services the ESB implementation lasts for about 1 year. It is not realistic to shorten this time without impacting quality of the implementation.

More time spent on early phases (analysis) shortens and simplifies the next phases. It is also good approach to implement ESB product in phases to verify that selected concept is valid and also for psychological reasons to see things going on.

The following figure shows a sample ESB implementation time plan with relationships between phases. The sample lasts for 11 months starting on beginning of June 2007.

![Figure 5: ESB implementation time plan](image)

8. Impacts of the implementation
ESB implementation (or SOA implementation) evokes need of changes in organizational structure, processes and other areas. If these changes are not properly handled the implementation may become inefficient and can have also another negative impacts. In following chapters main changes are being discussed with proposed ways to solve them.

8.1 IT governance
In SOA architecture (in this example implemented using ESB product) there is a huge focus on service (including other relevant topics) management.

This requirement is based on the fact that:
- There are many services in information system implemented
- Services change often and quickly
- Many processes rely on particular service
- There are complex relations between services themselves and services and processes
Services may exist in parallel in multiple versions

8.1.1 SOA repository

The most effective way to manage information about services is by using SOA repository. SOA repository is a tool for managing services and all related information. It contains metadata about all artifacts. Its main purposes of usage are

[RUBES]:
- Service description (functional description, technical description, availability, contact person, performance, production statistics, failures, …)
- Descriptions of relations (inter service relations, process to service relations, message transformation, message routing, data mapping…)
- Managing model of information system (model of the whole environment)

8.2 Application changes

Once the SOA architecture is adopted, further applications development has to be done using different paradigm. Analytics and developers have to primarily focus on reusability of application parts. Different parts of the application have different purpose for different target users. These services have standard interface so they (including analytics and developers) are becoming more independent on the other functionality of the original silo.

8.3 Organizational changes

The most crucial for successful and efficient SOA and ESB implementation is to perform the reorganization in appropriate time. Most of the companies have their organizational structure based on application view which leads to IT departments being split according to application silos. Those departments are responsible for the application from the presentation layer to data layer. This approach complicates or in worst scenario makes the reusability impossible.

Successful SOA adoption requires the responsibility reorganization focusing on services. It means that instead of vertical splitting of IT departments horizontal splitting should be applied. (See following figures.)
8.4 Change management

Change management for purpose of this article means process for approving and implementing the changes in information system. If applications are implemented as silos, changes to these applications are made independently on other applications. Every application usually has its own lifecycle which is limited only when changes interface to others. Once split into services, every service can have its own lifecycle which has to be well synchronized with other changes. To assure correct functionality, well established change management process has to be implemented. It includes:

- Approval of changes
- Synchronization in changes (including analysis, development, test and roll-out)
- Contingency planning

From technical view multiple service versions running in parallel have to be supported to decrease the complexity of relationships between services.

9. Implementation requirements achievement

This chapter summarizes if and how the expected results of implementing ESB product were achieved.

9.1 Managing complexity

The target for ESB implementation in this area was to decrease the complexity of the system and/or to easily manage the complex environment. This goal is achieved by implementing following:
9.1.1 Central hub
ESB acts as central hub (logical view, no matter if physically distributed). It is easier to manage an integration layer by using one application/system than managing many peer-to-peer interfaces.

9.1.2 Repository
By using SOA repository the complexity becomes manageable. Every interface has standard description, owner, relationships and other information structured.

9.1.3 Processes
Well defined processes such as change management, deployment and others bring order to application integration.

9.2 Lowering expenses and shortening application lifecycle
These targets are merged into one chapter as they can be usually achieved by same factors.

9.2.1 Different lifecycle
Different lifecycle for particular application functionality shortens the application lifecycle. Small change to service required by some business process has to be done only instead of implementing new version of the whole application. But in contradiction, deploying many individual changes increase costs by creating an overhead of managing roll-outs. These two factors have to be well balanced to achieve required decrease of costs.

9.2.2 Retesting
Well defined interfaces between services bring the possibility to detect all unwanted changes to other applications. Focusing on particular functionality (service) in testing phase greatly decreases expenses.

9.2.3 Well defined interfaces
Focusing on functionality itself instead of solving many issues regarding interfaces fasten the development process as well as decreases costs. All above mentioned factors shorten application lifecycle which is crucial for company’s competitiveness. This allows company to launch new products quickly. On the other hand some of the changes decrease the costs as well as some increase them. The overall result depends on certain application portfolio, company’s business and processes implemented.

10. Major ESB products evolution trends
ESB products market is new and quickly evolving with many things not yet standardized. This chapter mentions main evolution trends on ESB products market. Expected growth of the market appraised by [GARTEVO] follows:
By 2010, more than 80% all software infrastructure products will embed ESB technology or require ESB technology as a prerequisite (0.7 probability).

10.1 Achievement of standards

All vendors aim their effort on standards achievement. Basic standards like XML, JMS are supported by all the vendors, others like WS-Security, BPEL, XSL only by some of them.

Vendors that built their solution on old messaging products have the biggest lack in standards achievement (IBM, Tibco).

There is a clear trend to support all relevant standards and if some missing to create and approve them by standardization bodies like W3C, ANSI or ISO.

10.2 BPM support

Most of the ESB product vendors focus on implementing BPM support into their suites. BPM is the heart of the ESB; it allows business process flow definition using standard business process execution language (BPEL). Major vendors (BEA, Oracle) already implemented BPEL 1.1 into their suites, but there is still lack of BPEL 2.0 support. My expectation is that by 2008 major vendors (BEA, Oracle, Soniq and Sun) will support it.

10.3 MDA approach

There is a lack of model driven architecture (MDA) approach in ESB suites. Only Oracle partially supports it, others even haven’t started thinking about MDA. I am expecting big boom in this area in mid future (2010-2012).

11. Conclusion

The ESB product market is not yet developed enough but brings the possibility to gain many benefits when implementing in early phase.

To select the right product with clear vision is important for company’s competitiveness. But the crucial is not the product selection but its implementation where many mistakes can be done. To avoid such situation it is worth to have exact planning with all the necessary steps identified and defined.

Literature


Many companies are nowadays in situation with high expenses spent on their IT and long time-to-market with the applications which support their business. For that reason the companies plan the SOA adoption. To support the SOA paradigm an infrastructure for application integration is needed. This area can be covered by ESB products which in one package bring many features for application integration and management.

Once decided to implement ESB product, many steps have to be taken in right order to maximize the profit of the implementation.

This article describes how to select the right product and which features to focus on. It also describes the steps to be taken as well as implementation project time plan. Benefit expected as a result from the implementation such as lowering expenses, shortening time-to-market as well as IT management are being analyzed.

At the end main evolving trends in integration area are mentioned to support companies with their decision in selecting appropriate ESB product.