Implementing a Service-Oriented Architecture for Small and Medium Organisations

Pascal Bauler, Fernand Feltz, Nicolas Biri, Philippe Pinheiro

Department Informatics, Systems, Collaboration (ISC)
Centre de Recherche Public – Gabriel Lippmann
rue du Brill 41
4422 Belvaux, Luxembourg
{bauler, feltz, biri, pinheiro}@lippmann.lu

Abstract: This paper explains how we have designed a service-oriented architecture by combining and extending several open source environments. We show how the IT architecture of the National Family Benefits Fund in Luxembourg got systematically modernized by solving issues related to Enterprise Application Integration, before introducing concepts like Enterprise Service Bus which was extended to offer service orchestration and business workflow support. By taking advantage of Model Driven Software Design and by introducing a framework for adding new components to the service bus, technical burdens to get used to the new IT environment got minimized. The cost saving due to the open source sub-systems and the ease of use make the proposed architecture particularly interesting for small and medium organizations.

1 Introduction

Integration issues are topics that are mainly addressed in large companies or administrations. However the problems related to heterogeneous IT environments are also well known in Small and Medium Enterprises (SMEs) or Small and Medium Administrations (SMAs). In this paper we explain various research results we collected from projects with actors in Luxembourg and in particular the Luxembourg National Family Benefits Fund (Caisse Nationale des Prestations Familiales) (CNPF) and we propose an approach how to tackle integration problems in SMEs and SMAs with limited budgets.

Due to the decentralisation of the IT environments and the small size of the country, even central administration services in Luxembourg have to be considered as SMAs, with a headcount of about 100 full time equivalents.

In the following sections, we explain how the IT environment of the CNPF got modernized in a step-by-step approach. First of all, technical issues related to Enterprise Application Integration (EAI) got solved by integrating various heterogeneous and distributed IT solutions ([HW04] and [Li03]). In a second phase we designed an Enterprise Service Bus (ESB) offering standards based connectivity, integration, routing and transformation of data. ESBs may consist of distributed but interconnected sub-components, which tackle by design issues like high availability and reliability (see reference [Ch04] pages 7-10). The proposed ESB assures the technical integration of existing applications covering Mainframe applications, Document Management Systems, custom developed applications by means of various technologies and classical relational databases. In a third phase, a business-oriented integration was realized by adding business workflow support and by setting up a complete Service-Oriented Architecture (SOA). When moving towards a SOA, the enterprise wide IT architecture is reoriented towards a service-oriented approach. As specified in [Oe05], a SOA has a significant business impact by granting access to complete business functions and workflows through services, which may be distributed over the entire company or administration.

As the proposed solution is focused onto SMEs and SMAs, the ease of integration and usage of distributed services is crucial. In order to achieve this goal we use Model Driven Software Development techniques (MDSD) as defined in [SV05] and we propose a generic development framework to develop new ESB components. Dedicated meta-models and development artefacts hide all technical integration issues encountered when accessing the SOA or when making new services available.

The proposed solution is a starting point for new horizontal applications, which take advantage of existing distributed and heterogeneous services in order to offer new business functions.

2 Moving towards a Service-Oriented Architecture (SOA)

Based on the explanations of the previous section, we took a 3-step approach. First of all we started with a technical integration of existing applications. These integration modules were reused in a second phase as connectors which linkup existing applications to the service bus. Once these components stabilized, we integrated a workflow management system able to run business workflows. In the coming paragraphs we will discuss the various topics in detail.

2.1 Application integration and connectors

During the first project phase, we focused onto the question on how to retrieve and to inject data into existing applications running at our partner's site. A first inventory of existing applications resulted in a list of heterogeneous applications, using different technologies, design paradigms and offering different integration possibilities, which could be grouped into 4 categories:

- Mainframe applications based on ISAM[1] files
- A document management system with clearly documented APIs
- Classical Client/Server applications accessible through JDBC[2]
- Custom built applications accessible through DLLs, Web Services or monolithical WinDev[3] applications. Integration of WinDev applications is particularly tricky as this development environment is typically used in departmental environments where integration issues are of little importance.

The design goal in this first project phase was to work out technical solutions able to get read/write access to all relevant data. This goal could only be achieved by combining several technical approaches and by doing some reverse engineering on poorly documented components. At the end of this phase, we had reliable and standardized interfaces to all existing business applications running in production. To minimise development efforts, existing interfaces available as DLLs, or WinDev modules, were reused and encapsulated by means of C++ and Java technologies, in order to guarantee transaction safety. The total number of interactions and interfaces between the various sub-systems is (n*(n-1)/2), where n is the number of implied systems. In the concrete scenario of the Luxembourg National Family Benefits Fund, described in this document, the number of implied systems is 9 as shown on figure 1 (2 mainframe applications, 1 document management system, 1 JDBC based client/server application and 5 custom built systems), which results in up to 36 interfaces between the various sub-systems. To fight this complexity, the concept of ESB was introduced and offered a standardized way of integrating the various connectors.

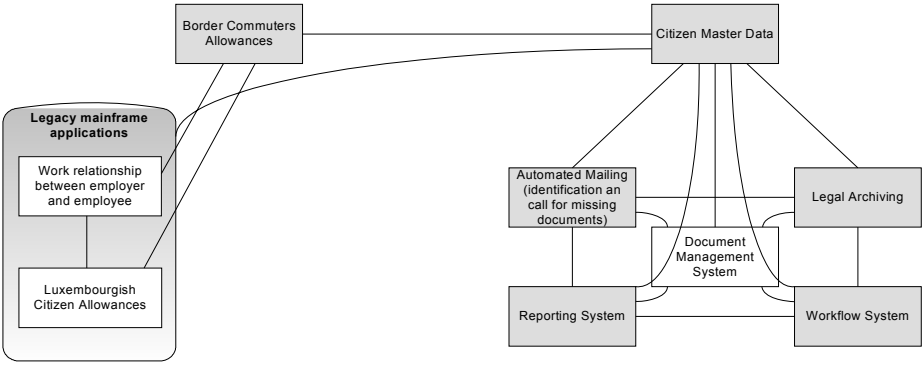

Figure 1. Initial system landscape at the CNPF

[1] ISAM: Indexed Sequential Access Method
[2] JDBC: Java Database Connectivity as defined by SUN Microsystems (http://java.sun.com/javase/technologies/database.jsp)
[3] WinDev 10 is a commercial rapid application development (RAD) environment conceived by PC-Soft (http://www.pcsoft.fr/windev/index.html)

2.2 Enterprise Service Bus (ESB)

The main benefit of this concept is its ability to handle all topics related to message routing, communication protocols and application integration in a central standardized way. Rather than establishing direct communication between EAI interfaces, or Web Services and a Workflow Management System as mentioned in [MKM05], we decided to introduce a Java Business Integration (JBI) [TW05] based ESB.

Indeed, the JBI specification and API define a platform for building enterprise-class ESBs using a pluggable, service-based design. The JBI runtime core mainly comprises the following components within the same Java Virtual Machine:

- Component framework enabling the deployment of the different types of components within the JBI runtime.

- Normalized Message Router [Vi05] enabling a standard mechanism of message interchange between the services. The messaging model uses the message exchange patterns defined in the Web Services Description Language (WSDL) [W306].

- Management framework based on Java Management Extensions (JMX) [KHW02] enabling the deployment, management and monitoring of components within the JBI runtime.

The JBI environment hosts plug-in components such as transport binding, routing engines, rule engines and transformation services. JBI defines two types of components: Service Engine Components (SE) responsible for implementing business logic and other services, Binding Components (BC) used to provide transport level bindings for the deployed services.

In order to deliver a service, each component needs a deployment package, called Service Unit (SU), containing deployment information. This information is composed of JBI standard information indicating consumed and produced services, as well as component specific information, proprietary for that particular component. Some complex components may require extra artefacts in order to provide a certain service (the Business Process Execution Language (BPEL) [An05] component for instance, relies on BPEL processes, which correspond to this type of artefacts). Such artefacts are packaged into the Service Unit.

In order to deploy SUs in a JBI container, they have to be grouped into packages offering composite services. These packages are called Service Assemblies (SA).

After thoroughly comparing the available solutions, we noticed that several stable open source solutions were available and, rather than developing a JBI container from scratch or buying a commercial solution, we decided to focus onto ServiceMix[4]. To guarantee platform independency only JBI standard functionalities were used and proprietary ServiceMix extensions were avoided. A consequence was the free choice of the JBI container to be used in the partner's IT environment.

The next step was to route all remote data accesses throughout this central system and, as a consequence, reduce the number of required interfaces. In this new design all components access the service bus through appropriate connectors or JBI Binding Components, when using the JBI terminology. A next step was to guarantee transport layer independency when accessing the various binding components. Dedicated transformation components were introduced, which act as access points for JBI client applications. These components accept connections using JMS[5], SOAP[6] or the Http Request protocol. The transformation components decompose incoming data and forward them to the appropriate components to access external applications. Any new client application has free choice to access all available services through any of the 3 above-mentioned protocols. In the current design we considered 2 types of binding components:

- Access points for external client applications that handle user requests, using any of the supported formats. Incoming requests are automatically transformed into appropriate JBI messages and forwarded to 'server side' binding components.

- Server side components accept JBI messages from the container and establish reliable data exchanges with external server applications.

[4] ServiceMix – an Open Source based *"Enterprise Service Bus"*, http://www.servicemix.org
[5] JMS: Java Message Service, a Java based framework for handling data exchanges through message queues
[6] SOAP: Simple Object Access Protocol, the underlying transport protocol of Web Services

All binding components required by our external partner (CNPF), have been realized and integrated into the above-mentioned environment. To guarantee platform independency, special care was taken to develop fully JBI compliant components without using any platform specific APIs. By doing so, we are free to deploy our solution and architecture in any JBI compliant IT landscape (as validated by replacing for test purposes ServiceMix by OpenESB[7]). The above-mentioned architecture offers a middleware acting as Enterprise Service Bus. By introducing the concept of transport protocol independency and a clear separation between client and server components, several technical issues became transparent and the complexity of using the proposed ESB got reduced. As all efforts have been based on Open Source technologies, initial costs of the ESB are minimal and the proposed solution is particularly interesting for SMEs and SMAs. Custom developed components have to be covered by a standard maintenance contract. Issues like high availability and geographical spreading of the service bus, which are handled by the JBI specification, were not relevant by now, as we concentrated onto SMEs and SMAs where similar constraints were not encountered. A next project phase was to extend this architecture into a SOA by making business workflows available throughout services.

2.3 Service-Oriented Architecture (SOA)

A prerequisite of running efficient business workflows is to add service orchestration to the JBI container. Functionality offered by several independent and distributed services are combined in business workflows orchestrated by the Workflow Management System. These business processes are made accessible as newly available services, available to the whole IT environment. When conceiving an orchestration service 4 key aspects were relevant for our project:

- Standards based Workflow management system.
- Existence of efficient and user-friendly modelling tools.
- Possibility to execute modelled workflows. As the workflows are supposed to be used in an existing operational IT environment, easy execution of the workflows is mandatory.
- Smooth integration into the JBI container.

[7] OpenESB– an Open Source based *"Enterprise Service Bus"*, https://open-esb.dev.java.net

After analysing the available standards and environments, we decided to aim for Business Process Execution Language (BPEL), which seems to be a very promising approach, with several implementations available. BPEL is specially suited for integration with JBI containers as both technologies rely on WSDL as service description language. After comparing several BPEL engines, ActiveBPEL[8] was retained and integrated. ActiveBPEL is an Open Source BPEL engine, which comes with user-friendly modelling tools. Compared to other free BPEL engines ActiveBPEL seems to be the most stabilized package. Full integration of ActiveBPEL as orchestration service into the JBI architecture required the usage of native ActiveBPEL APIs during the development process.

By definition, all communications with a BPEL engine are handled throughout web services using the SOAP protocol. By doing so, the BPEL engine is completely independent of the JBI container. This approach introduces a significant overhead, which is due to message transformations when exchanging data between the JBI container and the BPEL engine and vice-versa, and to network latency when initiating communication between JBI container and BPEL engine. To avoid these performance issues, a native integration of ActiveBPEL into ServiceMix was realized. As a consequence BPEL is able to access other JBI components directly by means of JBI messages. JBI components in charge of orchestrating communication between various other components are Service Engines (when using the JBI terminology). As there might be some controversy about allowing access to BPEL workflow throughout JBI by avoiding the SOAP transport protocol, we decided to make both options available. After integrating a BPEL engine, all technical pre-requisites for a SOA are fulfilled (figure 2 gives a detailed overview of the finalized SOA). To facilitate the administration of the SOA container, a JMX based monitoring tool was added. A Spring [Jo05] based graphical user interface was designed, to display JMX interfaces related to the JBI container and to the JBI components like ActiveBPEL and other custom developed components.

[8] ActiveBPEL – *"The Open Source BPEL Engine"*, http://www.activebpel.org

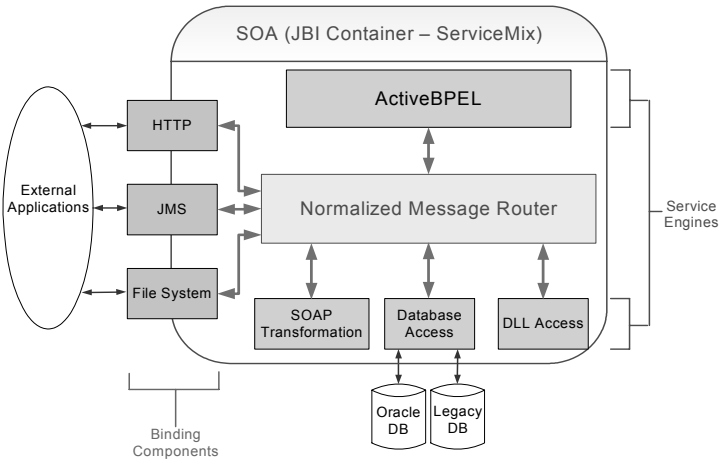

Figure 2. General overview of the proposed SOA

In the following sections we will briefly describe our efforts to facilitate access to the various services and the JBI container.

3 JBI component framework

According to the JBI specification, each component must implement the standard JBI API. After the development of a few JBI components, we noticed that many code segments were duplicated. In order to facilitate the development of new JBI components and to let the component developers concentrate onto the business logic, we proposed a common framework. This common framework eliminates code duplication and offers a coherent architecture for future JBI components. Furthermore, maintenance and improvement of the components are simplified. In this section we describe the proposed component framework and its practical usage by illustrating the design of two newly developed JBI Components. The selected components are the JMS transport binding component, taking care of routing/transporting JMS messages, and the "Report Factory" service engine, a generic reporting engine used to manage and execute queries on relational databases or document management systems. When explaining the various design choices, we frequently reference design patterns specified by the Gang of Four [Ga97].

The central element of the proposed framework, called the "component core", corresponds to the generic core features common to all JBI components. It is also in charge of coordinating the other elements of the framework. It takes advantage of the template method design pattern and it provides an abstract implementation of the JBI APIs mandatory for every JBI component. This component core guarantees that all components share the same basic behaviour towards the JBI container and component specificities can be added by extending this initial template.

Another key element of the framework is the “message handler”. This element handles incoming and outgoing messages transiting between the components and the JBI container. It provides a basic implementation of the message exchange patterns [W306]. This implementation verifies the validity of incoming messages and keeps track of the active exchanges involving the component. The message handler is developed according to the facade design pattern such that components can easily bind message reception to business functionalities. We may cite for instance the Report Factory component, where we can either add new queries or execute existing ones. These two operations correspond to business functions registered in the message handler. When a new message is received, the message handler checks its content to determine if a query must be stored or executed and call the appropriate function.

According to the JBI specification, each component requires a standard XML configuration file in order to be deployed in a JBI container and each valid Service Unit (defined in section 2.2) must contain an XML deployment descriptor. As the components are in charge of parsing those descriptors, a “configuration” element was added to the framework. The main role of this element is to parse the XML documents and convert them into valid Java objects. The JBI specification allows the extension of the XML configuration file. Such extensions can be used to tune specific aspects of the component. The “configuration” element parses the JBI standard configuration file and takes advantage of the decorator design pattern, which facilitates the parsing of component specific properties. For instance, in the JMS transport component, we need to define the receiving and sending queues of JMS messages. These properties are set in the component specific parts of the XML configuration files.

The JBI specification also defines a component management interface by means of JMX. There are two possible management levels, a standard one defined in JBI API (required) and a component specific one (optional). Consequently, a common standard JMX management interface is defined in the framework. This interface can be extended with component specific management functionalities. An example of component specific JMX management can be found in the Report Factory service engine. For this component we enable the dynamic parameterisation of the database connection, by specifying whether connections are pooled and by setting an optional pool size.

Finally, every JBI component must be able to deliver the metadata describing each service it provides, which corresponds to the WSDL service description. A common generator was defined at framework level. Each JBI component is free to add additional information to the generated WSDL by exploring the content of its JBI configuration files.

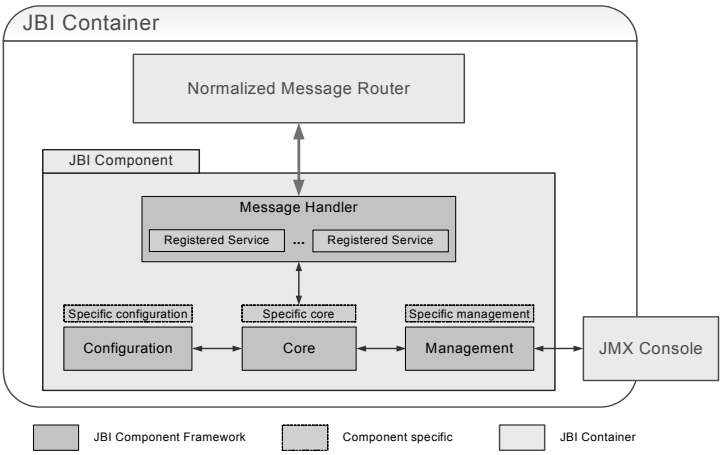

Figure 3. Architecture of a component built with the JBI Component framework

All these elements put together (as shown on figure 3) considerably reduce the development effort and technical complexity when conceiving new JBI compliant components. Furthermore, the proposed framework handles most of the JBI specific requirements and the development efforts are concentrated onto business specific requirements. Through several practical examples, we experimented the efficiency of the design patterns used in the proposed JBI component framework. It considerably facilitated the development of new JBI components and the integration of the proposed architecture in a large variety of integration scenarios.

4 SOA and Model Driven Software Development

When developing business applications taking advantage of a SOA, the source code can be split into two parts:

- *Business code*, realizing features like invoices, accounting information
- *Technical code*, handling problems like data persistency or access to the SOA container

Business code is usually domain-specific whereas technical code is more generic, containing a lot of repetitions and also requiring a detailed knowledge of underlying technologies. In order to overcome this problem we used a Model Driven Software Development (MDSD) approach to generate the technical code out of UML models. In practice the business application is modelled by means of UML diagrams. These diagrams are adjusted in order to fulfill the constraints imposed by the meta-model. When running the generators, the technical code is completely generated out of these UML models, as well as the skeleton of the business logic. Business developers complete the business logic manually by following appropriate design patterns like abstract factories and decorators [Ga97], in order to avoid problems related to protected regions in the generated code segments. The current meta model is specially fitted for EJB 3.0[9] and offers full integration with the JBI container. By adding specific tagged values and stereotypes into the UML models, you can automatically access remote and distributed services throughout the JBI container, or make business functions available as new web services. The code generator will generate the appropriate EJB code segments and annotations, JBI specific client- or server-oriented binding components as well as appropriate deployment information. As a consequence, SOA services can be transparently accessed and created by business logic developers and all technical complexity is hidden. OpenArchitectureWare[10], an open source framework, is used to offer MDSD functionality to the business logic development teams.

5 Use case at the Luxembourg National Family Benefits Fund

A first major horizontal application taking advantage of the SOA handles the payment of the family benefits in Luxembourg. Integration has to be established with the following sub-systems: a WinDev application managing master data, the mainframe holding administrative data, as well as a communication sub-system in charge of handling physical data exchanges with the French Family Allowances Agencies (CAF) and the banks to execute the payment orders. This application is currently in a testing phase and will be fully exploited in production beginning 2007 to handle the French border commuters.

[9] EJB: Enterprise Java Beans as specified in the JSR 220 http://java.sun.com/products/ejb/docs.html
[10] OpenArchitectureWare - *"the flexible open-source tool platform for model-driven software design"*, http://www.openarchitectureware.org

Major challenges when moving to a SOA are related to organisational issues and to a conceptual change in the perception of how to organize and optimize business processes. The Luxembourg National Family Benefits Fund, like many SMEs and SMAs, works on a paper-based approach, without having clearly defined business processes. Even projects like the installation of a Document Management System did not modify this initial concept, except that paper based files were replaced by scanned documents. As a consequence, a first task was to identify underlying business processes and to implement those processes in the SOA. During this Business Process Reengineering (BPR) phase, we used ARIS[11] EPC[12] as a communication tool between business and IT people. EPC was then used as basis for creating BPEL workflows, which on their side are too technical for business people. The business applications were modelled by means of UML respecting the constraints imposed by the meta-model around those BPEL processes. The code generator created the technical code segments (EJB, SOA and JBI related) hiding the technical complexity to the application developers. The new system landscape is detailed in figure 4.

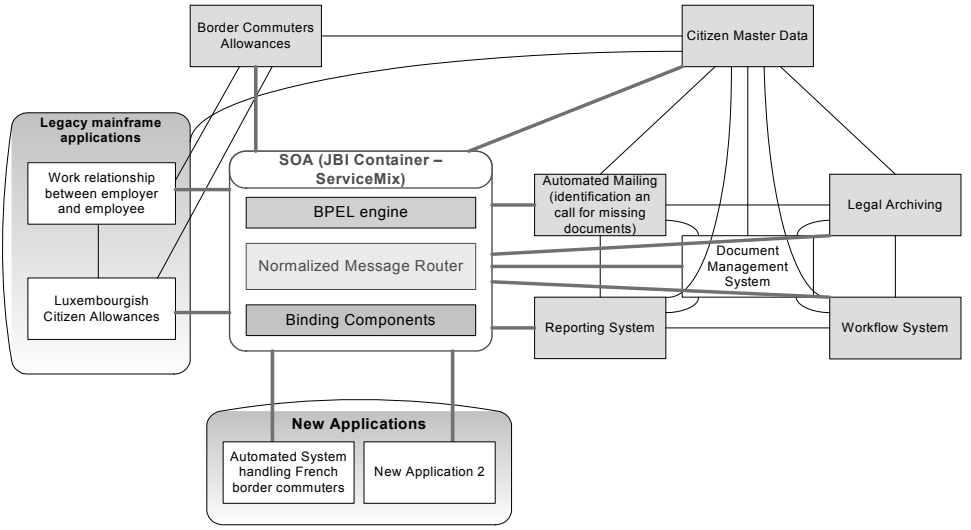

Figure 4. New system landscape at the CNPF

[11] ARIS: an modelling toolset provided by IDS-Scheer http://www.ids-scheer.com

[12] EPC: Event Driven Process Chain

6 Conclusion

In this document we give a rough overview on how we modernized the IT environment of the Luxembourg National Family Benefits Fund by introducing the concept of Service-Oriented Architecture. The proposed approach combines and extends several standards based open source solutions to systematically move from a EAI approach to the concept of Enterprise Service Bus respecting the JBI standard, to finally integrate service orchestration and business workflow support through a BPEL engine. We also show how this new IT environment gave life to new types of horizontal applications and how MDSD techniques facilitated the usage of this new environment.

Our contribution consists in realising the complete integration of the BPEL workflow engine into the JBI container as well as the design of a JBI development framework and a meta-model by means of MDSD techniques, in order to facilitate development of new and usage of existing JBI components. In addition, the management framework for the JBI container and BPEL engine were integrated and extended. This new environment was adapted to our partner's IT environment by developing adequate connectors and interfaces to existing business applications and by establishing a new development process, specially adapted to the Luxembourg National Family Benefits Fund.

The next step is to analyse how the SOA is deployed and help our external partner to take maximum advantage of the new possibilities. In parallel we will study other emerging integration initiatives like Service Component Architecture (SCA) [Be05], standards, which offer additional levels of platform and development language independency. We also plan to analyse how a higher level SOA architecture, as defined in [BGR05], can be integrated into our platform.

Acknowledgements

The authors want to thank Michel Neyens and Claude Nicolas of the 'Caisse Nationale des Prestations Familiales' for their support and collaboration in this project.

References

[An05] Andrews T., Curbera F., Dholakia H., Goland Y., Klein J., Leymann F., Liu K., Roller D., Smith D., Thatte S., Trickovic I., Weerawarana S., Business Process Execution Language for Web Services (BPEL4WS), ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf, May 2005.

[Be05] Beisiegel M. Blohm H., Booz D., Dubray J.-J., Edwards M., Flood B., Ge B., Hurley O., Kearns D., Lehmann M., Marino J. Nally M. Pavlik G., Rowley M., Sakala A., Sharp C., Tam K., Service Component Architecture – Assembly Model Specification (version 0.9), http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-sca/SCA_AssemblyModel_V09.pdf, November 2005.

[BGR05] Berbner R., Grollius T., Repp N., An approach for the Management of Service-oriented Architecture (SoA) based Application Systems, in Enterprise Modelling and Information Systems Architectures 2005 (EMISA 2005), p. 208-221, October 2005.

[Ch04] Chappel, D.A.: Enterprise Service Bus. O'Reilly Media, 2004.

[Ga97] Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley, 1997.

[HW04] Hohpe, G., Woolf, B., Enterprise Integration Patterns, Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, 2004.

[Jo05] Johnson R., Hoeller J., Arendsen A., Risberg T., Sampaleanu C., Professional Java Development with the Spring Framework, Wiley, 2005.

[KHW02] Kreger, H., Harold, W., Williamson, L., Java and JMX, Building Manageable Systems, Addison-Wesley, 2002.

[Li03] Linthicum, D.S., Enterprise Application Integration, Addison-Wesley, 2003.

[MKM05] Müller, J., Kümpel, A., Müller, P., Integration und Orchestrierung von Geschäftsprozessen in Web Applikationen – Ein Service-Orientierter Ansatz, in Entreprise Application Integration 2005 (EAI 2005), http://www.ceur-ws.org/Vol-141/paper10.pdf, july 2005.

[Oe05] Oey, K.J., Wagner, H., Rehbach, S., Bachmann, A., Mehr als alter Wein in neuen Schläuchen: Eine einführende Darstellung des Konzepts der serviceorientierten Architekturen, Enterprise Architekture. Unternehmensarchitekturen und Systemintegration, Band 3, Gito-Verlag, 2005.

[SV05] Stahl, T., Völter, M., Modellgetriebene Softwareentwicklung, Techniken, Engineering, Management, dpunkt.verlag, 2005.

[TW05] Ten-Hove, R., Walker P., Sun Microsystems, JSR 208: Java Business Integration (JBI), http://www.jcp.org/en/jsr/detail?id=208, final release, august 2005.

[Vi05] Vinoski, S., Towards Integration – Java Business Integration, IEEE Internet computing, http://dsonline.computer.org/portal/site/dsonline/menuitem.9ed3d9924aeb0dcd82ccc6716bbe36ec/index.jsp?pName=dso_level1&path=dsonline%2F0507&file=w4tow.xml&xsl=article.xsl, July 2005.

[W306] W3C, Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts, http://www.w3.org/TR/wsdl20-adjuncts, candidate recommendation, March 2006.