

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN



Exemplary Implementation of a Purchase Requisition Process according to the Principles of a Service-Oriented Architecture

Diploma Thesis (Diplomarbeit)

by cand. inform.

Oliver Dalferth

Eberhard-Karls-Universität Tübingen
Wilhelm-Schickard-Institut für Informatik
Arbeitsbereich Technische Informatik

Supervisors:

Universität Tübingen:

Prof. Dr.-Ing. Wilhelm G. Spruth

DaimlerChrysler AG:

Dipl. Wirt.-Inform. Michael Herrmann

Dipl.-Ing. Hans-Jürgen Groß

Day of Submission: September 1, 2007

Acknowledgements

This thesis was created in cooperation with the team ITP/AM Technology and Methods MCG of the DaimlerChrysler AG within the period between January, 2007 and July, 2007.

First, I would like to thank [Prof. Dr.-Ing. Wilhelm G. Spruth](#) for his advice, motivation and support. He guided the work to completion.

My special thanks go to my supervisor Michael Herrmann, DaimlerChrysler. I acknowledge his encouraging support, ideas, help and time which were essential for the project and the creation of this thesis.

Furthermore, I am grateful to Hans-Jürgen Groß, DaimlerChrysler for his trust and the fast and friendly integration in his team.

I thank Klaus-Dieter Jäger (LogicaCMG) for special and valuable advices and I appreciate the great help of Baghdad Abdessalam, Oliver Burgmaier and Franz Pieger (DaimlerChrysler).

I would also like to thank Hans-Juergen Brehm, Hermann Pauli, Werner Führich, Plamen Kiradjiev, Wolfram Andreas Richter, Andreas Konecny, Andreas Schmitz, Khirallah Birkler, Thomas Nold (IBM Deutschland GmbH), Dr. Dietmar Durek, Stefka Troyanova (IDS Scheer AG) and Andreas Suchert (SAP AG) for the great cooperation.

In particular, I am very grateful to my parents and my grandmother who always supported, motivated and helped me during my studies.

I am also grateful to Ingrid Schaumann for her understanding for busy days when I was working for this thesis.

Executive Summary

In order to illustrate the top-down approach within a Service-Oriented Architecture (SOA), a prototype was developed in cooperation with IBM Deutschland GmbH, IDS Scheer AG and SAP AG. A real world purchase requisition process at DaimlerChrysler AG was chosen for the prototype. Tools and services of different vendors were used for the project involving efforts of the vendors to improve compatibility among each other. The implementation of the purchase requisition process was done in several steps. First, a business process model was created using extended Event-driven Process Chains (eEPCs). The model was then converted into BPEL-code (Business Process Execution Language). Subsequently, the resulting BPEL was used for creating the IT implementation of the process using WebSphere Integration Developer. Finally, the process was deployed on the WebSphere Process Server. There was no need for an Enterprise Service Bus (ESB) for the prototype.

The resulting working project demonstrated that processes can indeed be implemented with this SOA approach. The prototype proved to be convincing and benefits and drawbacks of Service-Oriented Architectures could be analyzed based on the project.

Especially, the process was modified in order to demonstrate the flexibility and agility that can be achieved with a SOA. An ad hoc implementation of a process change was demonstrated. The process modification was carried out based on the top-down approach starting with a change of the business process model.

Software of different vendors (IDS Scheer AG, IBM and SAP AG) was used for the process implementation. This proved the possibility of combining heterogeneous tools and services in a SOA. WSDL (Web Services Description Language) interfaces were developed for external SAP applications in order to enable their successful integration.

During the project, an insufficient state of maturity of SOA development software was recognized. Compatibility problems among software of different vendors were found and caused implementation problems. A solution of these compatibility problems was developed in close cooperation with the vendors. Because of the immaturity of the existing BPEL standard, vendors developed proprietary BPEL extensions and compatibility problems were based on these vendor specific differences.

Contents

1	Introduction	2
2	SOA - Basics	4
2.1	SOA	4
2.1.1	Definition of SOA	4
2.1.2	Historical Context	5
2.1.3	Motivation for SOA, Expected Benefits and Drawbacks	7
2.1.4	Principles of Service-orientation	10
2.2	The Term Service	12
2.2.1	Functions, Classes, Objects, Components and Modules	12
2.2.2	Service Definition	12
2.2.3	Granularity	13
2.2.4	Composition	14
2.2.5	Orchestration and Choreography	14
2.3	Business Services	15
2.4	Web Services	16
2.4.1	Definition	16
2.4.2	Description	16
2.4.3	WSDL	17
2.4.4	SOAP	17
2.5	Service Call	19
2.5.1	The “Find, Bind, and Execute Paradigm”	19
2.5.2	Service Calls in a SOA Landscape	20
2.6	SOA Lifecycle	22
2.7	The Enterprise Service Bus	23
2.7.1	Definition of an ESB	23
2.7.2	Connection	23
2.7.3	Protocol Independence and Pattern Support	24
2.7.4	Transport	25
2.7.5	Mediation	25
2.7.6	Security	25
2.7.7	Implementation of an ESB	26
2.8	BPEL	27
2.8.1	Business Process	27
2.8.2	A BPEL Introduction	27
2.8.3	BPEL4People	33
3	Strategy for Top-Down SOA Projects	37
3.1	Overview	37

3.2	Business View	39
3.2.1	The Business Process Model	39
3.2.2	Event-driven Process Chains (EPCs)	39
3.2.3	Identifying Services or Service Candidates	45
3.2.4	Designing the IT Version of the Business Process Model	46
3.3	IT View	49
3.3.1	BPEL Conversion	49
3.3.2	Completing the Process	49
4	Practical Example (SOA Live Session Project)	51
4.1	Motivation	51
4.2	Overview	52
4.3	Business View	55
4.3.1	Assessment	55
4.3.2	The Business Process Model	58
4.3.3	Identifying Services or Service Candidates	60
4.3.4	Designing the IT Version of the Business Process Model	61
4.4	IT View	65
4.4.1	BPEL Conversion	65
4.4.2	Completing the Process	71
4.5	Graphical User Interfaces	80
4.6	Monitoring	86
5	Change Request	87
5.1	Strategy	87
5.2	Practical Example	88
6	Summary and Conclusion	95
6.1	Results of the SOA Live Session Project	95
6.2	Conclusion	96
	List of Figures	97
	List of Tables	99
	Listings	100
	Acronyms	101
	Bibliography	103
	Appendix A: Process Descriptions	108
	Appendix B: ARIS Rules for Event-driven Process Chains	113
	Appendix C: CD Content	116

1 Introduction

This thesis is based on the need for an example project built according to the principles of a *Service-Oriented Architecture* (SOA). A better understanding of SOA should be provided and benefits and drawbacks should be explored. A prototype¹ was implemented at DaimlerChrysler AG following the top-down approach. The process was implemented based on a business process model and software solutions of different vendors (IDS Scheer AG, IBM, SAP AG) were used. Not only local services on the *WebSphere Process Server* but also external applications on a *SAP Discovery System* were involved.

In order to demonstrate the flexibility and agility that can be reached in a service-oriented environment, the business process was modified and re-implemented. The advantage of fast changing processes in a SOA was shown.

In chapter 2, the Service-Oriented Architecture, its properties, components and predecessors are introduced. Several relevant terms are defined and also an introduction is given into the *Business Process Execution Language* (BPEL). BPEL and the extension *BPEL4People* [III⁺05] is presented because of its importance for SOA projects.

Chapter 3 describes a general strategy for top-down SOA projects. Different views (business view/IT view) are distinguished in this chapter depending on the tasks that have to be performed. Process modeling with *Event-driven Process Chains* (EPCs) is explained which belongs to the tasks of business unit members of a company. Their detailed process knowledge and understanding are required for modeling processes the right way. A created model cannot be directly used for an IT implementation as it first has to be converted into a format that can be read. Before a conversion is possible, the model has to fulfill several rules that are required by the converter. An IT version of a business process model has to be created that keeps all these rules. The help of an IT specialist might be necessary depending on the expertise and experience of the modeler. The chapter also covers the IT view of project development in a SOA. The process can be implemented based on the model and different possible implementation solutions are explained.

In chapter 4, the introduced strategy is applied to a practical project (SOA Live Session project). The top-down implementation of the prototype is illustrated. A created business process model is converted into BPEL-code and a running project is created and deployed on a server. All steps are explained in detail and practical experiences of the

¹a prototype for demonstration purposes

prototype are described. Also the creation of a Graphical User Interface is described in this chapter and a short introduction in process monitoring is given.

Chapter 5 explains how to deal with process modifications and also provides a practical example related to the prototype.

Chapter 6 covers the results of the project. Positive and negative experiences are described. As the prototype was presented to the executive management at Daimler-Chrysler AG, feedback and future developments are discussed.

2 SOA - Basics

2.1 SOA

2.1.1 Definition of SOA

A *Service-Oriented Architecture* (SOA) represents a software architecture concept that is not based on a specific technology. The concept is based on the way software is implemented. Finding and publishing services that describe certain business functions and the interaction with services represent central issues of a SOA. A variety of different definitions can be found in literature covering different aspects of SOA. Diverse perspectives on software architecture that are influenced by business or technologically driven views are the reason for the big differences. First, three dissimilar definitions that shall help demonstrating these perspectives are exemplarily mentioned starting with a business view [BBF⁺06].

“A set of business, process, organizational, governance, and technical methods to reduce or eliminate frustrations with IT and to quantifiably measure the business value of IT while creating an agile business environment for competitive advantage” [BBF⁺06].

Also definitions exist that are based on the approach of implementing a Service-Oriented Architecture using Web services. The following definition refers to that technology:

“Contemporary SOA represents an open, extensible, federated, composable architecture that promotes services-orientation and is comprised of autonomous, QoS-capable, vendor diverse, interoperable, discoverable, and potentially reusable services, implemented as Web services” [Erl05].

A technically even more concrete description specially focuses on services in general and includes already a short service definition. It already refers to concepts that will be described later in this thesis:

“A Service-Oriented Architecture is a software architecture that is based on the key concepts of an application frontend, service, service repository, and service bus. A service consists of a contract, one or more interfaces, and an implementation” [KBS05].

Another definition of the World Wide Web Consortium (W3C) focuses on the expression “components” and their interface descriptions:

“[A Service-Oriented Architecture is] a set of components which can be invoked, and whose interface descriptions can be published and discovered” [OWRB06].

Also Gartner developed an own description:

“SOA is a software architecture that builds a topology of interfaces, interface implementations and interface calls. SOA is a relationship of services and service consumers, both software modules large enough to represent a complete business function. So, SOA is about reuse, encapsulation, interfaces, and ultimately, agility” [OWRB06].

As the project for this thesis was developed at DaimlerChrysler AG, also the interpretation of the team for technology and methods (ITP/AM Technology and Methods MCG) shall be presented where services and their properties are focused:

“SOA is a technology neutral architectural concept based on generally (re-)usable services. This concept of software architecture represents one or more business functions as a service. The interface of a service is platform independent. The implementations of the services are reusable, encapsulated and loosely coupled. The service interactions are realized by a standardized/uniformed infrastructure” [Dai07].

2.1.2 Historical Context

Before the development and usage of systems based on the *Client-Server model* [Har04], *host-terminal communication* [Ghe97] represented the used IT technique on the market. In a host-terminal-system, one powerful central computer (mainframe) serves multiple terminals that just connect to this system in order to use it and to get the computed results. The terminals themselves are just I/O-devices that don't provide any computing power or memory. As the computing power demands can't be spread throughout the network but have to be served by one instance, a central dependency of one single mainframe computer exists in host-terminal systems. The host serves the different terminals by implementing a *time-sharing system* [CG00] where the computing time is divided in time slices that can be allocated to the different programs that have to be run. This way, many clients can be served at the same time and each client gets the impression of having the full computing power at one's disposal. Whereas the necessity of time-sharing is considered obvious today, it was a revolutionary approach in the late 50ies [Ghe97] [CG00].

The clients in a Client-Server model can be equipped with more intelligence than terminals. Still the clients connect to one or several servers but tasks like error management for instance can already be addressed by the clients. It is intended to discharge the server system by spreading the work. Often applications with graphical user interfaces run on the clients which send requests to servers and wait for the response. The Client-Server approach is wide-spread and many file, mail and application servers are installed and used privately or by companies [Har04].

In order to reach a better separation of tasks and a higher level of abstraction *Three-Tier* [SCD00] approaches developed where the functionalities are separated more strictly.

It is intended to implement the user interface, the functional (business) logic and the data storage as different modules. The modules are ideally implemented on different platforms. Similar to the Client-Server Architecture, the user interface is implemented on the client whereas the business logic and the data storage systems are located but separated on the server side. This way, modules can be replaced by other implementations independently without carrying too much weight to other parts of the Three-Tier approach. The fact that modules can also be reused in other systems leads to a higher grade of flexibility. Another advantage of this architecture in comparison to the Two-Tier Model is an increase of performance and scalability. A better balance concerning the assignment of tasks exists there [SCD00].

Also architectures with more tiers are possible (n-tier, multi-tier) trying to separate the individual tasks even better. With the demand for good performance serving a huge amount of clients, distributed systems like the *Distributed Computing Environment* (DCE) [KBS05] and CORBA (Common Object Request Broker Architecture) [KBS05] arose. The DCE was developed in the early 90ies and used the *DCE/RPC* (Remote Procedure Call). CORBA was influenced by the growing popularity of object orientation. Objects communicate in CORBA over an *Object Request Broker* (ORB) [KBS05] and are able to manage their own state. Stubs on the client side and skeletons on the server side communicate and serve as proxies for clients and servers [OMG07]. Abstraction is provided by the ORB as knowledge about the objects' location is not necessary. Using a common Interface Definition Language (IDL) [KBS05] makes it possible to let objects communicate that are implemented in various programming languages. Although CORBA seemed to be a sophisticated solution for creating a distributed environment, it turned out to be too complex for enterprises and their need for software reuse [KBS05].

Enterprise Java Beans (EJB) [KBS05] introduced by Sun Microsystems in 1997 are based on the concept of clustering a set of objects into a single server. With a controlled number of servers hosting the objects, EJB proved to be able to manage the limited number with its supported transaction management, naming services and security [KBS05].

But even more middleware solutions like the *X/open-based CORBA Object Transaction Service*, the *Microsoft Transaction Server*, the *Java Transaction Service*, *CORBA Notification*, *Java Message Service* (JMS) and *Enterprise Application Integration* (EAI) [KBS05] developed and created a middleware heterogeneity. Middleware that was originally trying to solve the problem of application heterogeneity created therefore compatibility problems at a higher abstraction layer. The *Extensible Markup Language* (XML) [KBS05] should help to address the communication problems and was developed in the mid 1990ies. *SOAP* (first name: Simple Object Access Protocol, later: Service-Oriented Architecture Protocol or just SOAP) [KBS05] was created for transporting the XML-messages over *HTTP* (Hypertext Transfer Protocol) [KBS05]. Today, current implementations of SOAP are independent of the underlying messaging transport mechanism. However, by implementing a protocol based on HTTP, all the work

that has already been done concerning security standards, load balancing, failover and application management for HTTP could be reused. The existing infrastructure for browser-to-server communication could this way be the basis for server-to-server communication. An interface definition language called *WSDL* (Web Service Description Language) [KBS05] was later developed by Microsoft for SOAP services [KBS05]. With the new implementations of WSDL and SOAP, a new approach for addressing the problem of middleware heterogeneity was started.

The development of the information technology always experiences a lot of changes. New approaches develop for both programming languages and network techniques and the evolutions often seem to be completely new and confusing for people that are not yet familiar with them. Looking closer on the new designs, one can experience that the concepts didn't actually change but a new abstraction layer was built on top of existing layers in order to be able to manage more complex scenarios. The current layer is represented by Web services and the Service-Oriented Architecture. Implementing a Service-Oriented Architecture with Web services is one proposal for solving this problem [DJMZ05]. The reduction of IT complexity is intended by developing in a SOA.

2.1.3 Motivation for SOA, Expected Benefits and Drawbacks

Reacting fast to the changes of market, consumer behaviour and therefore business requirements is a serious and important aim for enterprises. Flexibility is very attractive and it is needed for staying competitive on the free market. The implementation of a SOA within a company is expected to increase the capability of adapting the IT landscape to changes and reducing complexity. IT systems should be capable to be integrated more flexible with a SOA. This way, the necessary agility that the market requires, a better time to market and an optimization of business processes shall be reached. An improvement of the IT efficiency and a lowering of arising costs by consolidating IT systems belong to the expected benefits of SOA. SOA also promises the support of reuse concerning business processes and value added chains.

A considerable amount of time and money is spent by enterprises in order to achieve fast and flexible IT systems but the price for developing a SOA in a company is expected to be rewarded soon by savings concerning the easier integration of future changes to the IT landscape. Therefore, in the end a lowering of IT costs shall be aimed. A lower price for IT changes also implies a lower risk for an enterprise.

Vendors emphasize the following drivers for the service-oriented approach [KAH⁺05]:

1. The speed of changing existing products and processes or recombining them in new ways is expected to be increased. Also new implementations shall be finished faster.
2. Costs for implementations and ownerships of IT systems and their integration shall be reduced.

3. Outsourcing of business elements in a more fine-grained way than previously possible shall enable flexible pricing models. The movement from fixed to variable pricing based on transaction volumes shall also enforce this flexibility.
4. Integration work required for mergers and acquisitions shall be simplified.
5. A better return on investment and IT use shall be achieved.
6. Application and platform independence shall be reached for the implementation of business processes.

Vendors also underline the following advantages for the loosely coupled and flexible integration of IT systems in a SOA [KAH⁺05]:

1. As interfaces describing services are implementation-independent, an integration of heterogeneous systems can be enabled.
2. Interdependencies are minimized to just business relevant issues by the description of service interfaces with the help of terms of a common business process and data model.
3. Encapsulating services with standard interfaces enables their reuse and flexibility. Service changes are straightforward as services are supposed to be defined and implemented in only one specific place.

Also the advantage of having different lifecycle speeds for each service should be mentioned (figure 2.1). Various independent technologies can be connected in a service-oriented approach and an independence of the way services are implemented exists. Technological skills are allocated at the implementation part of services and hidden for the service caller. As a number of services belong to a process, the process has to be identified and analyzed well before it can be determined definitely. This procedure leads to a better understanding and visibility of processes and therefore to a better maintainability. As services can be replaced by others and as there is no dependence on the service location, outsourcing and offshoring can be carried out. The possibility of fast service replacement supports a gradual migration to newer technologies. Costs are spread across projects as modules can be developed independently and also maintenance costs reduce [Pez06].

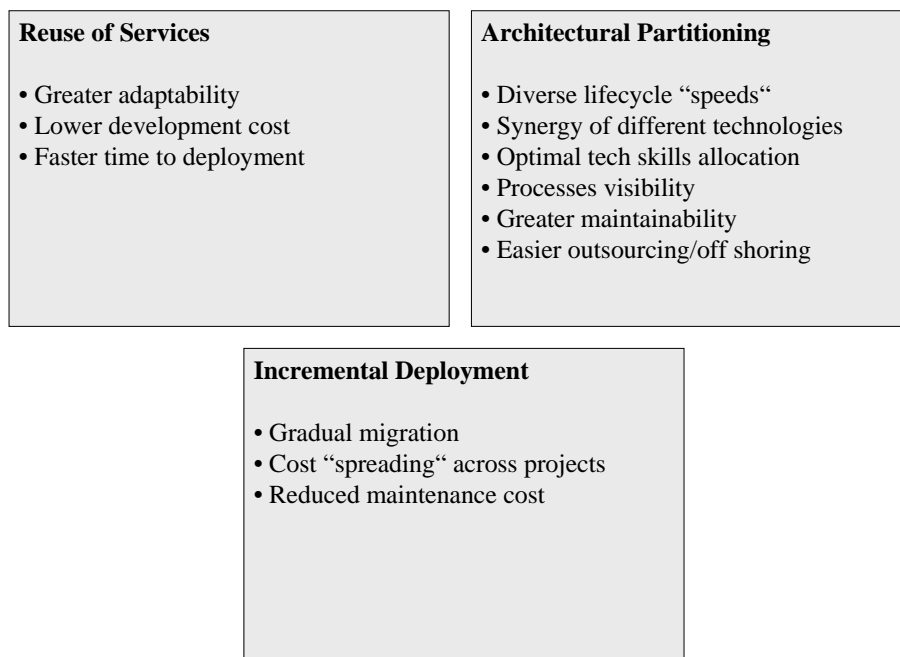


Figure 2.1: Expected benefits of SOA [Pez06]

But also drawbacks concerning a Service-Oriented Architecture should be mentioned (figure 2.2). This architecture means a new way of designing, developing and implementing in an IT landscape and therefore a cultural change. Engineers have to get used to the new way of thinking. It takes more formal methodology and a longer development time for services that are meant to be reusable because of the fact that questions about functionalities and the level of granularity have to be answered [Pez06].

Higher upfront costs are the result of the difficult establishment of an *Enterprise Service Bus* (ESB) and the individual design addressing the local requirements of an enterprise. Justifying the price for creating such an infrastructure will take a lot of effort. The more distributed infrastructure enforces the extensive use of middleware and brings up a lot of difficulties in testing, debugging, troubleshooting, metering, logging, security and transaction management [Pez06].

Also governance problems come up in such a new landscape. Questions about the service ownership, the accountability and the cost allocation have to be answered. Conventions about prioritization of service requests have to be established and conflicts arise if service requests cannot fulfill all defined prerequisites of a service [Pez06].

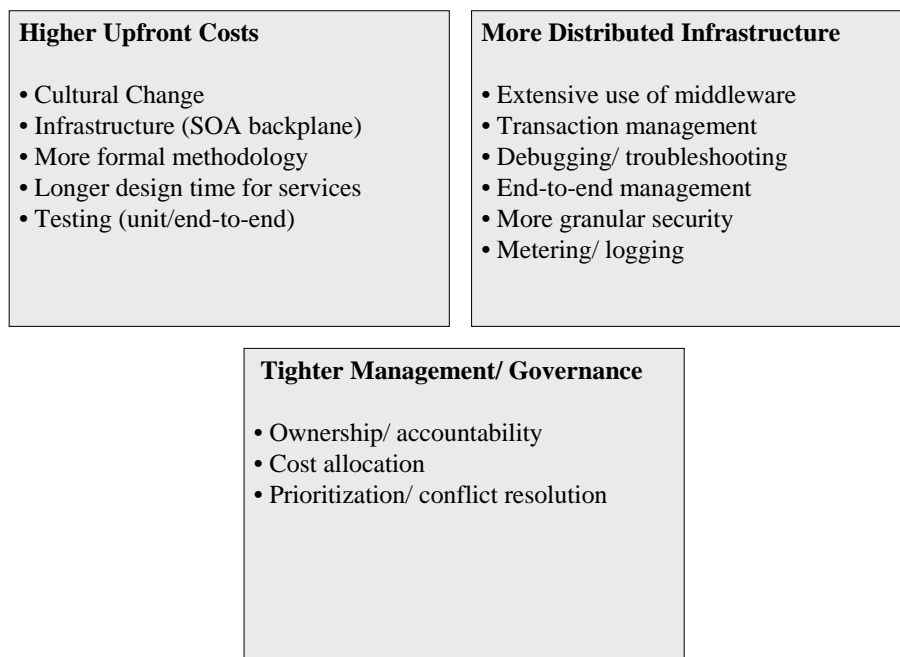


Figure 2.2: Expected drawbacks of SOA [Pez06]

2.1.4 Principles of Service-orientation

Encapsulation, flexible coupling and the reuse of software components (services) in new contexts belong to the main aspects for a SOA. With SOA, an agile runtime environment shall be created. The definition of service interfaces are crucial for SOA. The conceptual ideas behind a SOA are characterized by certain principles with special demands for services [Erl05] [GYVN03]:

1. **Loose coupling**

Dependencies between services are minimized. Only an awareness of each other shall be retained. The aspect of loose coupling differentiates SOA from the modularity of basic software.

2. **Service contract**

One or more service descriptions and other related documents collectively define a communications agreement belonging to a service.

3. **Autonomy**

The encapsulated logic is controlled by the service itself. The implementation software of a service has its own architecture. Sufficient service isolation is necessary for avoiding the creation of a monolithic application.

4. **Abstraction**

Besides descriptions of the service contract, logic is hidden by services from the outside world following the black box principle.

5. Reusability

Reuse of services is enabled by dividing and spreading logic across different services.

6. Composability

Composite services can be formed by coordinated and assembled collections of services. Integration of services is required in order to bundle and manage all the independent services. Use, tools and skills define the optimal granularity of services.

7. Statelessness

The retaining information is minimized expressing only a specific service activity.

8. Discoverability

Services are designed in a way that enables an outward description so that they can be found and called with the help of available discovery mechanisms.

But Gartner also notes that “SOA is not always the right architecture [GYVN03]”. However, “runtime SOA is evident in dynamically reusable business-scope software components ” [GYVN03].

2.2 The Term Service

This section shall help to differentiate between the definition of services and other terms like functions, classes, objects, components or modules.

2.2.1 Functions, Classes, Objects, Components and Modules

Functions are subroutines representing program blocks with an own name [SW01]. They can accept values for calculations or other work and they can pass back return values.

Classes are usually used in the context of object-orientation. “In the software model, a class is a piece of program text that describes the fundamental properties of the objects it can create. A class is defined by its name, its inheritance relationship to its super-classes and a set of object properties. [...] These properties include the interface of objects and their internal implementation by algorithms and data structures” [Zül04]. A class therefore “defines the creation and behaviour model of its instances” [Zül04]. Functions of classes are also called methods and they define the behaviour of objects. “An *object* is always an instance of exactly one class” [Zül04]. It owns an own identity and an own dataset. It represents a unit of data and functions that operate on the data. The structure of data and functions of uniform objects are defined in their common class [SW01].

“A software *component* is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party” [Szy02].

A *module* is an enclosed functional unit with complex performance and can be exchanged without affecting the rest system [Kla06]. To meet the requirements of a component, a module has to provide specified interfaces.

2.2.2 Service Definition

A service is represented by any discrete function which can be offered for the usage by an external consumer. The function must not necessarily be an individual business function but can also be implemented as a collection of functions building a process [KAH⁺05]. The service is accessible by a platform independent interface, its implementation is hidden behind a Black Box and its state is not externally observable. The state is encapsulated and can only be communicated by providing a callable function giving information about the state. The service is a unit that is provided by a third party and can be called and reused. However it has to be deployed only once [Sie05] [GSM02]. In literature, services are also called “components” because of their well-defined interfaces and their deployment independence.

2.2.3 Granularity

The granularity of services is an important aspect for the implementation of a SOA as the communication overhead and traffic amount of an IT infrastructure depends on it. Several separate things can be associated with service granularity in a SOA [KAH⁺05]:

1. Service abstraction level (basic granularity):
The service abstraction level differentiates the different types of services. A service can be represented by a very low-level technical function, a sub-process or activity at a lower level or a high-level business process.
2. Service operation granularity (functional granularity):
The service operation granularity describes the number of operations included by a service. Factors have to be discussed determining which operations are covered by a service.
3. Service parameter granularity (interface granularity):
The number and the type of input and output data of service operations has to be addressed and a small number of large, structured parameters are preferred to a small number of primitive types.

The performance of the resulting system has to be taken into account when determining the type of granularity for certain functionalities. Too many interactions between endpoints are required once too fine-grained services are used. On the other hand, too coarse-grained services can involve large information exchanges that might not be necessary. A Poor interface design can be the result of an interface creation for a service where little work is done with each message, based on a message content model that is not suitable for complex, multipart messages. The best way of avoiding such poor interface designs is to use an information model that dynamically adapts to the consumer's needs and a rather coarse-grained interface. Especially for external consumption, a coarse-grained interface granularity is recommended. As an example, a coarse-grained interface could be used for a complete processing of a service [BBF⁺06] [Col04].

Fine-grained interfaces might be used inside an enterprise as more flexibility is granted for requesting applications. However, interaction patterns may vary between different service requesters and the support of a service provider can be made more difficult using fine-grained interfaces [Col04].

Coarse-grained interfaces guarantee a consistent usage of services. They are not required but recommended in a SOA as a best practice for external integration. With the help of service choreography a coarse-grained interface can be created running a business process that consists of several fine-grained operations [Col04].

2.2.4 Composition

Services should be designed in a way that they can be used in different contexts. A major property of a SOA is the ability of developing and modifying processes dynamically based on existing services. Service composition corresponds to the creation of new services from existing ones. This way, reusable services are transformed into a new single service fulfilling the task of a certain business functionality. If the needed business requirements change, the service composition can be modified or a new one can be established using other services [WM06]. Composability therefore corresponds to another form of service reuse. An appropriate level of granularity is needed for maximizing composition opportunities and also the design of service operations has to conform to a common standard for this reason [Erl05].

2.2.5 Orchestration and Choreography

For building service compositions, services have to be combined the right way. Service orchestration and choreography is necessary for managing the combination and interaction of services. In order to differentiate the two terms, both ideas are explained and figure 2.3 demonstrates the difference:

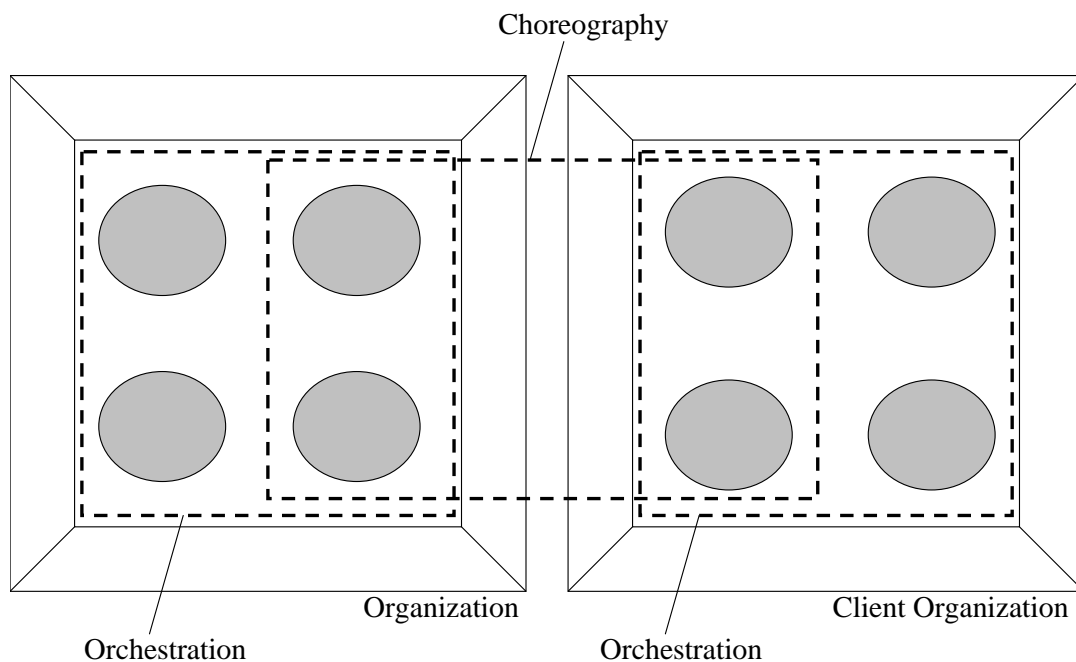


Figure 2.3: Orchestration And Choreography [Erl05]

The interpretation of orchestration is not unique in service-oriented environments. Orchestration is used in order to express the business process logic via services. Business

logic can be represented and expressed in a standardized, services-based way through the use of orchestration. An organization-specific business workflow is expressed by an orchestration. The logic is owned and controlled by an organization even if interaction with external business partners is involved [Erl05].

An orchestration is defined by the way one Web service invokes other Web services concerning sequences and conditions. A certain useful function shall be realized with the help of an orchestration. An orchestration corresponds to the pattern of interactions that a Web services agent (system) has to follow for getting a desired result [HB04].

A Choreography represents a community interchange pattern that is used by services from different provider entities for collaborative purposes. It is not necessarily owned by a single entity. Requirements for organizations to interoperate with the help of services are becoming more and more real and complex. The need for collaboration requires multiple services belonging to different organizations to work together [Erl05]. A choreography is therefore defined by the way multiple cooperating independent agents exchange messages for performing a task [HB04].

2.3 Business Services

A business service is a service that represents one or more business functions. It implements a meaningful business process or task [GG06]. A set of services that are assembled under a common rubric building a logical grouping of services are represented by a business service [OAS04].

2.4 Web Services

As Web services were developed and used for the thesis project, a short introduction is given in this thesis. However, Web services and the related protocols are not the focus of this thesis. For further information please refer to [WCL⁺05], [Erl05] and [Nüb07].

2.4.1 Definition

There are various definitions and descriptions that often describe Web services as loose coupled components that communicate platform independently via Web standards. Other more restrictive definitions refer to distributed applications that were developed with the help of the specifications of SOAP, WSDL and *UDDI* (Universal Description, Discovery and Integration) . As SOAP and WSDL play a decisive role for Web services and as the W3C has managed the evolution of their specifications, the Web service definition of the W3C shall be cited in this thesis:

“A software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with XML serialization in conjunction with other Web-related standards” [WCL⁺05].

2.4.2 Description

Although Web services are often mentioned when talking about Service-Oriented Architectures, SOA stands for an abstract architectural concept in contrast to Web services that only represent one approach of realizing a SOA. SOA is based on loosely coupled components (not necessarily Web services) that can be composed and discovered and that are described in a uniform way [WCL⁺05].

However, Web services have again motivated the industry to accept the challenge to find a uniform way of describing, locating and accessing components or services in a distributed environment. The aspect of loose coupling represents the essential difference between Web services and other traditional approaches like CORBA or *DCOM* (Distributed Component Object Model) . In contrast to collections of objects or components that are well understood at development time but that are tightly integrated, the Web service approach is characterized by dynamics and adaption to changes. Another key difference is the fact that Web service technology and specifications are developed in an open way. Consortia such as the *Organization for the Advancement of Structured Information Standards* (OASIS) or the W3C, industry partnerships and standards for common technology used for the foundation of the internet, support the development of the Web service approach [WCL⁺05].

2.4.3 WSDL

Metadata that fully describes services and their characteristics are defined by service descriptions. Loose coupling in a SOA can only be reached if an abstract definition of necessary information for deploying and interacting with a service is given. The Web Services Description Language represents a special XML vocabulary in order to describe Web services. Service authors can provide decisive information about services with the help of WSDL in order to enable the use of the service by consumers. A particular property of WSDL is its extensibility and adaptability. It enables the description of services using different type systems like *XML Schema*, *Java* or *RelaxNG*. Also various different protocols like SOAP, RMI/IIOP (Remote Method Invocation over Internet Inter-ORB Protocol) or in-memory calls can be specified for the communication with a service. A reusable abstract part and a concrete description belong to a WSDL document. The abstract definitions describe the operational behaviour of a service. Messages that go in and out from Web services are recounted for getting operational information. The concrete descriptions in a WSDL tell the caller where and how to access a service implementation [WCL⁺05].

2.4.4 SOAP

A relatively lightweight and simple mechanism for the exchange of structured and typed information between Web services is provided by SOAP. *Simple Object Access Protocol* was the first name for SOAP and later it was called the *Service-Oriented Architecture Protocol*. In order to dissociate from both descriptions it is just called SOAP today. SOAP represents an underpinning for Web services. The development of SOAP is based on the intention to reduce costs and complexity of application integrations on heterogeneous platforms. An extensible enveloping mechanism is defined by SOAP and message exchanges between Web services shall be structured with help of SOAP [WCL⁺05]. SOAP messages are XML documents that contain three different types of elements. An *envelope* represents the root element which again contains the other element types *header* and *body*. Extensible features can be added with help of the optional header element. A header element contains header blocks and SOAP defines attributes that can be used to indicate who should deal with which block and whether it is mandatory or optional to deal with it [WCL⁺05]. The container for the payload in a SOAP message is represented by the body element. There is only one payload allowed which is also used for error reporting. The body element is always the last child element of an envelope and contains the actual message content. No built-in header blocks are defined by SOAP. Today, SOAP allows many different alternative messaging transport mechanisms because of the fact that SOAP is independent from transports for message exchange. However, as HTTP is one of the first protocols that were included within the SOAP specification, most Web services of the first generation communicate using HTTP. But in general, the transport independence allows a very flexible routing. Even a change of the transport mechanism between routing nodes is possible. SOAP also provides a message flexibility which allows services to communicate based on a variety of message exchange patterns. This way, the diversity of

distributed applications can be satisfied [WCL⁺05].

SOAP also supports different message formats. The *Document* [FM02] and the *RPC* format [FM02] represent the most important ones. The documents-style message format represents the default format in most development kits as it is considered to be more flexible than the RPC-style format. All features of the RPC format and even more are covered by the document counterpart. For the document format, a Web service method has only one argument that is sent as a real XML encoded object. Every possible XML document can be sent [FM02].

The RPC format uses Web services corresponding to remote procedure calls. A Web service method can own parameters and return parameters and the values have to be recoded. Type information for each attribute have to be defined in contrast to the document-style format where type information is rather included in the `<types>` section of WSDL documents [FM02].

When it comes to the encoding of messages, bigger differences between the formats stand out. *Literal encoding* is usually used with the document style format and is based on the fact that body contents have to conform to a specific XML Schema. *SOAP encoding* uses a set of rules that are based on the XML Schema datatypes for the encoding. However the message does not conform to a specific schema. SOAP encoding is used with RPC-style formats [FM02]

2.5 Service Call

2.5.1 The “Find, Bind, and Execute Paradigm”

In a SOA, the communication between service consumer and service provider is done according to the “find, bind, and execute paradigm” which is also called the “SOA-paradigm”. Here, the service registry is used to enable a connection between the service requester and the service provider (figure 2.4): The service consumer sends a request for a special required service to the service registry. If the service is registered, the registry answers the consumer by sending the endpoint address for the requested service and a service contract. Otherwise a response is sent including a message telling that the service is unknown. If the service was found, it can be called by the consumer with the received address information and the contract. The service provider allows the service consumer to execute the service according to the received contract.

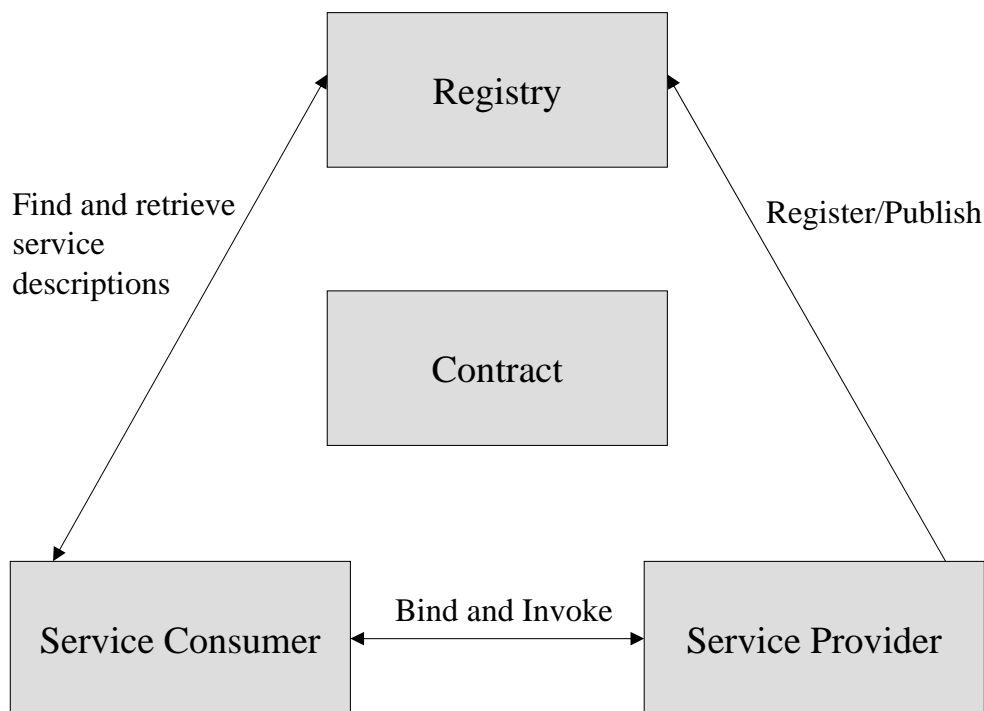


Figure 2.4: The Find, Bind, and Execute Paradigm [MTSM03]

Four core entities cooperate in a SOA to realize the Find-Bind-Execute-Paradigm:

Service Consumer

A service consumer can be any software module like an application or a service. The consumer requires a special service and initiates the service that is located in the registry. It binds to the service and executes the ser-

vice function. The service execution is done by sending a request which is formatted according to the contract [MTSM03].

Service Provider

A service provider represents the service that accepts and executes requests from consumers. It can be any software system like a mainframe system or a component that is addressable over a network and that executes service requests. It publishes its contract in the service registry for enabling access by consumers [MTSM03].

Service Registry

A service registry represents a network-based directory containing information about available services. Contracts from service providers are accepted, stored and provided for interested service consumers. [MTSM03]

Service Contract

The way of interaction between provider and consumer is specified in the service contract. Specifications about the format of the service request and the response are provided. Preconditions and postconditions might be required that describe the state of the service when executing a particular function. Also *Quality of Service* (QoS) levels might be specified. QoS levels represent specifications for nonfunctional aspects of a service like the required amount of execution time of a service method [MTSM03].

2.5.2 Service Calls in a SOA Landscape

Figure 2.5 shows the big picture that was developed at DaimlerChrysler AG for demonstrating a possible SOA landscape. All different applications that were developed by using different kinds of systems provide services that are described with the help of WSDLs. Service information can be found in one or different kinds of registries and repositories. The implementation of the repository connection is still in the fledgling stages. All the different systems can be accessed over an Enterprise Service Bus which is explained in section 2.7 starting on page 23. The Business Process Execution Language is used to call and combine the different services. A Graphical User Interface (GUI) can be created with the help of *JavaServer Faces* (JSF) for a concise graphical demonstration of processes. On the top level, *Value Chain Diagrams* (VCD) and extended Event-driven Process Chains are used to model business processes.

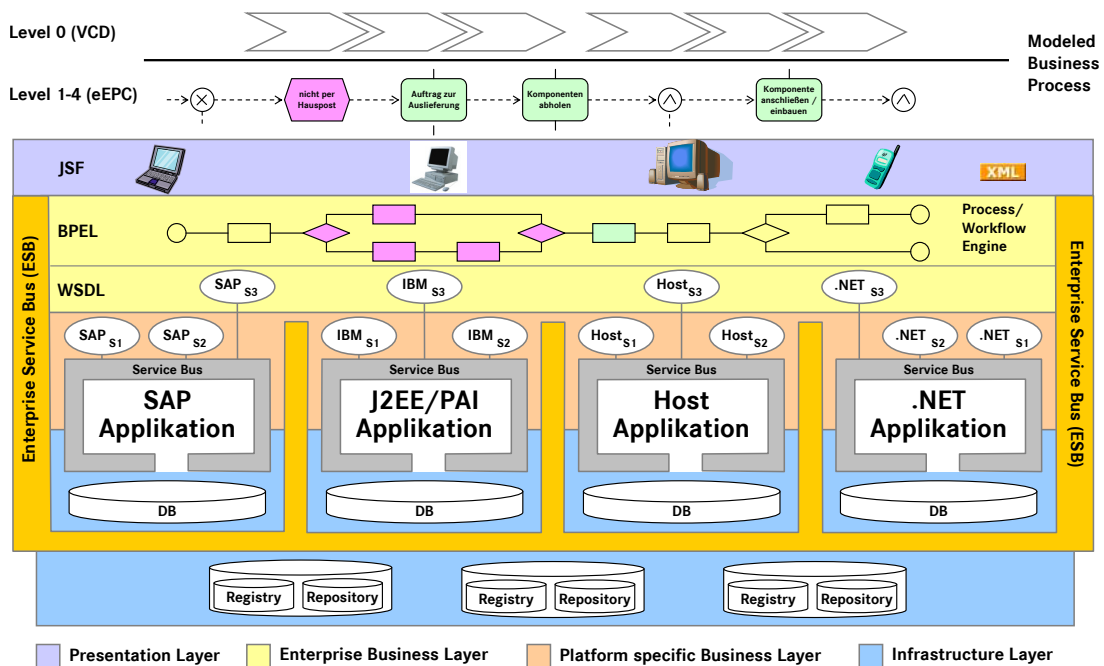


Figure 2.5: Big Picture SOA [Dai07]

Services are registered and looked for in the registry. The service registry contains information about the services like service contracts, policies, interfaces, operations, parameters and the actual location of services [KAH⁺05]. Meta information about the services is stored in the repository which is implemented as a robust framework. It is designed to be extensible to suit the varying nature of service usage. Relevant information for a repository could be logging, auditing, QoS information or ontologies describing the service interface semantically. After finding an adequate service that fits the needs of a service consumer, the necessary information about the service is retrieved from the registry and repository and a connection between the service consumer and the service provider can be established [KAH⁺05].

2.6 SOA Lifecycle

A SOA does not correspond to a static architecture which is implemented once and that does not experience any changes. It is a dynamic one and the process of development in a SOA is compared to a lifecycle. First of all, a business process has to be modeled with all its properties. However, the process model has to be translated in an information system design in order to make it runnable. The process can be deployed on a server after its implementation offering a service that can be called and used. Based on results of service calls, the process can be refined and missing process steps can be added to the business model. Monitoring helps finding process steps in need of improvement and the lifecycle can restart. The whole cycle repeats until a stable business process implementation is reached. As a business process might lose its importance after a certain time, it might be undeployed or replaced by a different one. If the old process implementation is still used, versioning should be taken into account.

As an example, IBM's SOA strategy is characterized by the SOA lifecycle that is defined by the terms "Model, Assemble, Deploy and Manage" and tools were developed for every step of the lifecycle. The life-cycle tools were developed for the realization of the described top-down approach and IBM provides detailed information for every step and tool. See [DRS⁺07] for more information about IBM's SOA lifecycle.

2.7 The Enterprise Service Bus

This chapter introduces the ESB and explains various important properties and aspects. For more information please refer to [KAH⁺05] and [BBF⁺06].

2.7.1 Definition of an ESB

“Service interactions are realized by a standardized/uniformed infrastructure” [Dai07] called the *Enterprise Service Bus* (ESB). The essential evolutions of the last years concerning IT architectures like Enterprise Application Integration, Web services and SOAs are all based on changes of the IT systems perspective. Equal integration concepts are used by the ESB but a focus on data flow exists [DJMZ05].

For the realization of a self-managed, automated SOA, the ESB represents an essential architectural element [BBF⁺06]. An ESB represents a core intermediary tying services together into logical, componentized sets. Minimal heterogeneity concerning the business semantics that are exposed by services is ensured by the logical grouping and design of services. A facade is formed by every service for the hidden implementations of the business logic [BBF⁺06].

The central task of an ESB is the exchange of data between IT systems or parts of them. The data exchange happens in the form of service calls. However, no direct connection exists between the service consumer and the service provider and routing and transformation capabilities have to be provided.

2.7.2 Connection

As an intermediate component, an ESB should support transparency concerning the connection to any service with its proprietary implementation using special data types and programming languages. Therefore, a big challenge is posed for an ESB. Various systems that do not directly support service-style interactions shall be linked by an ESB in order to offer a variety of services in a heterogeneous environment.

An ESB in a service-oriented environment avoids any direct connection between service consumers and providers. The service consumers establishes a connection to the ESB and not to the service provider who offers the actual service. A decoupling of the service consumers from the providers takes place [KAH⁺05]. Service requestors and providers can be located anywhere in a distributed environment [BBF⁺06]. Also services that are located externally of an enterprise can be reached and integrated. Business functions in components can be invoked without the need of regarding special protocols or application interfaces by using services that are defined by a standard interface description (WSDL) [BBF⁺06].

Figure 2.6 demonstrates the ESB and different possible component types connected to it. Applications, diverse data, orchestrated and other services in a distributed en-

enterprise computing infrastructure are connected with the help of an ESB. The ESB therefore represents an intelligent, distributed, transactional, and messaging layer for establishing this connection [BBF⁺06].

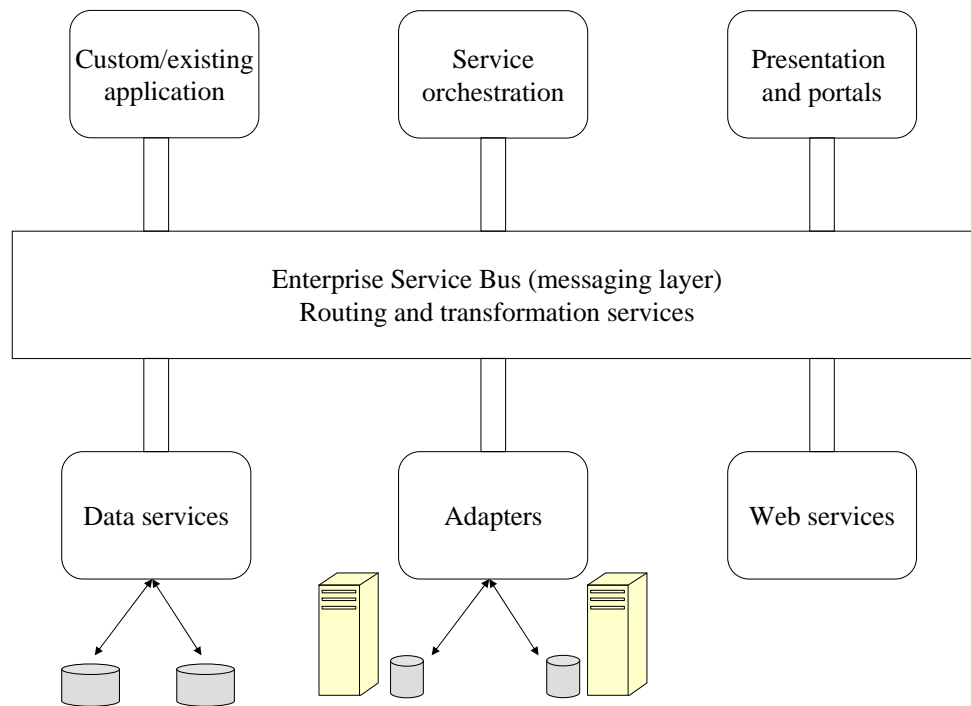


Figure 2.6: Enterprise Service Bus and connected component types[Bal05]

2.7.3 Protocol Independence and Pattern Support

An ESB provides intelligent content-based routing (i.e.: QoS-based), mediation, transport and gateway services which all are essential for a SOA. Also transformation capabilities ensure a reliable message passing using an ESB. An ESB hides the connection details of how to connect to a specific provider from a consumer. Without the ESB, the consumer would have to connect directly to the provider. The consumer would have to know and use the protocol, transport and interaction pattern that the service provider uses. The ESB though corresponds to an intermediary that passes requests from a consumer to a provider and manages the use of all the appropriate protocols and patterns needed by the provider. Several integration mechanisms have to be provided by an ESB in order to make specific transformations and data conversions possible. Various patterns like *request/response*, *publish/subscribe* and *events* must be supported in one infrastructure. SOAs with all their reusable services but also message-driven and event-driven architectures must be able to communicate via an ESB. Messages that are sent and received by applications must be supported. Also events that are generated and consumed independently of other applications have to be processed [KAH⁺05].

2.7.4 Transport

For Web services, essential transport protocols are SOAP over HTTP/HTTPS or SOAP over JMS but an ESB should also be able to connect to applications using other protocols like RMI over IIOP. With the help of adapters like *Java 2 Connector Architecture* (J2C or JCA) even more applications shall be callable like applications using the *System Network Architecture* (SNA) protocol or base TCP/IP, for example. [BBF⁺06]

2.7.5 Mediation

The enabling of an intelligent processing of service requests, responses, messages and events represents the mediation in an ESB. An implementation of mediations can be located at service endpoints (either service consumer or provider) or halfway between both participants implemented through the ESB infrastructure. In the second case they represent true intermediary components. They operate on logical SOAP message representations sent between service consumers and providers. Mediation handlers have to process the header of SOAP messages but SOAP processing and routing is not the only purpose mediations can be used for [BBF⁺06].

A mediation covers transformations like translations from XML to a different type of XML, aggregations and database lookups. Message validation is possible by the verification of any data field or a combination of several fields following specific rules. A selection of services based on QoS or content is also covered by a mediation. As an example, a customer with high priority should probably be able to call services with a higher throughput than other customers. Country information in service parameters could be essential for routing activities.

The mediation is justifiably used for logging and auditing of service interactions. Anchor points for manageability should be provided by an ESB for metering and monitoring in order to control the behaviour of services and the ESB itself. A mediation should implement an autonomic behaviour and react when special events for self-configuring, optimizing, healing etc. are detected. Externalized policies based on XML should address the behaviour of all the mentioned aspects [BBF⁺06].

2.7.6 Security

Security properties like encryption services, delivery assurance and other aspects that concern a reliable connection are supposed to be covered by an ESB. This way, applications are discharged and do not have to address all these problems anymore [KAH⁺05]. Flexibility and manageability of enterprises is improved if proprietary service implementations don't have to be touched for the management and the integration of services [KAH⁺05]. The maintenance of the integrity and confidentiality of the connected services should be ensured by an ESB. Existing security infrastructures for addressing authentication, identification, access control, data integrity and confidentiality should be integrated with [KAH⁺05]. The access control method *Single-*

Sign-On matters for ESBs where a user should only authenticate once but gain access to resources of multiple software systems [BM03]. Also incident reporting, disaster recovery and contingency planning belong to these aspects. An overall security management, monitoring and administration has to be covered. An ESB can either integrate with existing security solutions or implement them directly [KAH⁺05]. Please refer to [Nüb07] for more information about security in a service-oriented environment.

2.7.7 Implementation of an ESB

As an architectural construct, the ESB can be designed in various ways. The ESB can be a central server following the ideas of the *hub and spoke model* meaning that the ESB represents the hub and the different paths to it are considered spokes [Shi03]. The ESB can also be a complete decentralized implementation or a federate system (a connection of several central systems) [Dai07]. It can be implemented by the usage of classical messaging, brokering technologies, EAI or by using platform-specific components like service integration buses in J2EE systems (i.e.: the WebSphere Application Server). Also combinations of EAI and application server technologies are possible. However, the overall architecture should not be affected by the final implementation. For the choice of possible ESB implementations, an architecture assessment is necessary where existing IT infrastructures, processes and skills are analyzed and evaluated [BBF⁺06].

2.8 BPEL

2.8.1 Business Process

Often the term *business process* is associated with operations that have to be done in an insurance company or a bank. This is why sequences of activities that are performed by various persons are often called a business process. These activities are typically repeated and follow the same pattern, the *process model*. But also assembly-line productions of cars or the different phases in software development represent a typical example for a business process. Also activities with minimal user interaction can be described as business processes like implemented computer processes that just have to be started but that might involve a lot of different steps. Therefore, all batch jobs can be interpreted as business processes as well. The notion of business processes covers a really wide spectrum and it is the business of the user that determines what a business process is. [LR00]

2.8.2 A BPEL Introduction

With the help of WSDL, standards and specifications have been developed for the creation of an infrastructure that enables the creation of services in a network, their discovery and the interaction with services. But there also exists a necessity for the definition of logic over a set of service interactions. For this purpose the *Business Process Execution Language* (BPEL) was developed with its first name *BPEL4WS* (BPEL for Web services) . Later the name was changed to *WS-BPEL* (Web Services BPEL) commonly known as just *BPEL* [WCL⁺05]. BPEL as an imperative programming language is layered on top of the XML specifications WSDL 1.1, XML Schema 1.0 and XPath 1.0 (XML Path Language) . The used data model is provided by WSDL messages and XML Schema type definitions [ACG⁺03]. Choreography data can be accessed and manipulated by using XPath. With BPEL, a set of services can be composed and also an interaction protocol of a single service can be defined. The interaction protocol can be created by the specification of an ordering of the services' operations. BPEL is an extensible workflow-based language and it aggregates services by the choreography of service interactions. The aggregated services might be used again in other choreographies involving a recursive aggregation. As BPEL is decoupled from the choreographed service instances, a highly dynamic environment is created where services can be substituted or changed. On top of the Web service specification stack, BPEL composes services and defines service interactions by using their WSDL interfaces. BPEL offers a big modularity and allows to add further aspects to the BPEL-code like the attachment of Quality of Service policies or the copy and exchange of endpoints of composed services (using WS-Addressing endpoint references) [WCL⁺05].

WSDL descriptions represent the foundation for the description of business processes in BPEL. A built BPEL process is again represented and accessible as a regular Web service. BPEL differs between the specification of abstract business protocols and executable processes that both share a common amount of language elements that rep-

resent most of BPEL. However, in contrast to abstract business protocols, executable processes need additional information like endpoint addresses of the Web services that take part in the communication. Every BPEL document includes the following elements (listing 2.1) [Dje04]:

Listing 2.1: BPEL process [Dje04]

```
1 <process name = "ProcessName">
2     <partnerLinks>...</partnerLinks>
3     <variables>...</variables>
4     <correlationSets>...</correlationSets>
5     <faultHandlers>...</faultHandlers>
6     <compensationHandler>...</compensationHandler>
7     <eventHandlers>...</eventHandlers>
8     activity
9 </process>
```

The `<process>` element names the process and all its child elements are optional except activities that are represented by base activities or structural activities. Base activities are atomic operations providing elementary effects like the communication with other services and consist of the elements

receive, reply, invoke, assign, throw, terminate, wait,
empty, scope and compensate

The structural elements

sequence, switch, while, pick and flow

describe orders and conditions of encapsulated elements that again can be base or structural activities. Structural activities allow the composition of base activities. BPEL also provides a construct with the name `<scope>` that allows the definition of nested activities with all the elements that also belong to the root element `<process>` except the `<partnerLinks>` element. This way, variables can be defined locally or exception handling can be restricted to certain scopes. Scopes correspond to bracket terms in imperative programming languages. By using `<partnerLinks>` communication types can be created and role names can be assigned. The specification of one role is sufficient for one-way communications. Two roles are necessary if instances call each other involving both-way requirements for portTypes. The instances of portTypes correspond to partnerLinks and their attributes assign actual Web services to declared roles. Concrete endpoints have to be assigned to roles for executable processes but the assignment can depend on proprietary BPEL implementations. Listing 2.2 illustrates a partnerLinkType with two roles where the actual process is represented by the role "Buyer" [Dje04].

Listing 2.2: Definition of `partnerLinkTypes` [Dje04]

```

1 <partnerLinkType name ="BuyerSellerLink">
2     <role name ="Buyer">
3         <portType name ="BuyerPT"/>
4     </role>
5     <role name ="Seller">
6         <portType name ="SellerPT"/>
7     </role>
8 </partnerLinkType>
9
10 <partnerLinks>
11     <partnerLink name ="buying" partnerLinkType ="BuyerSellerLink"
12         myRole ="Buyer" partnerRole ="Seller"/>
13 </partnerLinks>

```

All possible `partnerLinks` can be combined to `partners`. This way, different roles are possible at different times for the Web services of a business process. For example, the role “Seller” could be replaced by the role “Shipper” at a later time for a shop [Dje04]:

Listing 2.3: The `<partners>` [Dje04] element

```

1 <partners>
2     <partner name ="Shop">
3         <partnerLink name ="Seller"/>
4         <partnerLink name ="Shipper"/>
5     </partner>
6 </partners>

```

Operations that are offered by other Web services can be called with the `invoke` element (listing 2.4). Name information about the operation, `partnerLinks` and `portTypes` are needed for referencing the remote operation that has to be called. Eventual arguments or return variables are specified by setting *message types* for `inputVariable` and `outputVariable`. With message types any number of nested variables can be defined and sent [Dje04].

Listing 2.4: Invoking the operation “buy” with the input variable “itemid” [Dje04]

```

1 <invoke partnerLink ="buying" portType ="SellerPT" operation ="buy"
2     inputVariable ="itemid" outputVariable ="response"/>

```

As the process also represents a regular Web service, also own operations can be offered. With help of the `<receive>` element (listing 2.5) an operation and information about `partnerLink`, `portType` and `variable` can be specified. The attribute `createInstance` defines if new instances of the process are created for each incoming Web service call. In this example a process can be started by calling the operation

“buy” with `portType` “SellerPT” and by handing over an “itemid” [Dje04].

Listing 2.5: The `<receive>` element [Dje04]

```
1 <receive partnerLink ="selling" portType ="SellerPT" operation ="buy"  
2     variable ="itemid" createInstance ="yes"/>
```

For synchronous operations, the calling service expects a reply value or an error reply. Return parameters can be specified with the `<reply>` element (listing 2.6) for that purpose. Synchronous operations are completed with this element [Dje04].

Listing 2.6: With the `<reply>` element a price is returned to the caller [Dje04]

```
1 <reply partnerLink ="selling" portType ="SellerPT" operation ="buy"  
2     variable ="price"/>
```

Sequences in BPEL enable a sequential order of activities. An activity is only executed in a sequence if all predecessors have been already executed. The sequence in listing 2.7 demonstrates a synchronous Web service call [Dje04].

Listing 2.7: Sequence for a synchronous Web service call [Dje04]

```
1 <sequence>  
2     <!-- activity 1 -->  
3     <!-- activity 2 -->  
4 </sequence>
```

The flow element enables parallel execution of encapsulated activities. It supports concurrency but also synchronization as synchronization dependencies can be specified [ACG⁺03]. The `<link>` element declares `link` names that can be used as precondition or effect for other activities. If an operation A shall be the predecessor of an operation B, a `link` can be defined and set as source for operation A and target for operation B (listing 2.8). An activity with a target element is only executed after the execution of the activity with the corresponding source element [Dje04].

Listing 2.8: Flow example for executing A before B [Dje04]

```

1 <flow>
2     <links>
3         <link name ="AtoB">
4     </links>
5
6     <!-- invoke operation A -->
7     <invoke ..>
8         <source linkName ="AtoB"/>
9     </invoke>
10
11    <sequence>
12        <receive ../>
13        <!-- invoke operation B -->
14        <invoke ..>
15            <target linkName ="AtoB"/>
16        </invoke>
17    </sequence>
18 </flow >

```

An activity can also be the target of several sources with the help of a `join` condition. With the operation `getLinkStatus` also more complex synchronization dependencies can be defined [Dje04].

Variables can be defined that can be set to message types, XML schema types or XML schema elements (listing 2.9).

Listing 2.9: Variable definition to an `xsd` type `int` [Dje04]

```

1 <variables>
2     <variable name ="itemid" type ="xsd:int"/>
3 </variables>

```

But variables cannot only be assigned return values but also fixed values with the help of an `<assign>` element (listing 2.10).

Listing 2.10: Assigning a value to a variable [Dje04]

```

1 <assign>
2     <copy>
3         <from>50</from>
4         <to variable="maxprice"/>
5     </copy>
6 </assign>

```

Business processes communicate via messages with usually invisible content. However, in some cases this information might be necessary for the further execution of

a business process and this is why access to special internal variable contents is enabled with `<property>` elements (listing 2.11). An alias has to be defined in order to assign a property to an attribute and the alias can be accessed in order to affect the actual attribute. The property of a variable can be read by executing the operation `getVariableProperty('variableName', 'propertyName')` [Dje04].

Listing 2.11: Defining a property for enabling public access to an attribute [Dje04]

```

1 <property name="userID" type="xsd:string"/>
2 <propertyAlias propertyName="userID" messageType="orderDetails"
3   part="identification" query="/credentials"/>

```

With `<correlationSets>` an amount of properties can be defined that uniquely identify a Web service in order to ensure that correct sessions receive the intended messages. A `correlationSet` with the name “userid” is created in listing 2.12 in order to identify users [Dje04].

Listing 2.12: Defining a `correlationSet` for the user identification [Dje04]

```

1 <correlationSets>
2   <correlationSet name="userid" properties="username, userpw"/>
3 </correlationSets>

```

The sets can be used in combination with communication activities like `invoke`, `receive`, `reply`, `onMessage` and event handling. By encapsulating the element `<correlations>`, messages are sent to the correct identified receiver. An `initiate` attribute defines if the `correlationSet` properties are set to the values of the incoming message or if the message values are interpreted to send the message to the correct local process instance [Dje04].

Several other elements also belong to BPEL but are only introduced shortly. Please refer to [ACG⁺03] and [Dje04] for further information. The `<switch>` element enables case differentiation and the `<pick>` element offers the use of different exclusive `<onMessage>` elements. Depending on the incoming message, special activities can be started. Also while loops are possible with the element `<while>` and a special condition attribute. A blocking `<wait>` element stops the program flow for a denoted time and the instance of an executable business process can be terminated by `<terminate>`. An `<empty>` activity can be defined for syntactical reasons. BPEL also addresses exception handling with its elements `<throw>`, `<catch>`, `<catchall>`, `<terminate>`, its `faultHandler` and its `compensationHandler`. Also `<event-Handlers>` can be defined for special incoming operation calls or timeouts, for example [Dje04].

Lately, the new specification *WS BPEL Version 2.0* [JE07] released but for the implementation of the project of this thesis, WS-BPEL 1.1 was used as WS-BPEL 2.0 was not yet fully supported by used software.

2.8.3 BPEL4People

WS-BPEL primarily focuses on automated business processes that orchestrate activities of multiple Web services and on their observable behaviour. However, also human user interactions are required in practice by many business processes as people also take part in business processes and influence their execution. New aspects like human interaction patterns have to be addressed. Simple scenarios such as a manual approval but also complex ones involving the entering of user data must be enabled. SAP and IBM developed a joint white paper with the name *BPEL4People* [III⁺05] in order to include the missing human interaction support in WS-BPEL. Scenarios for involving users in business processes have been described and extensions to the current WS-BPEL have been addressed. BPEL4People is based on BPEL in order to make it possible to compose BPEL core features with the new features of BPEL4People. Different scenarios illustrate the scope of BPEL4People [III⁺05]:

People Activities

People take part in business processes as a new implementation of an activity in BPEL. This new version may be called *people activity* and can be understood as an assigned task to a user. “A task is an indivisible unit of work performed by a human being” [III⁺05]. Particular users may be specified at design time, deployment time or at runtime. For the specification at runtime, organizational directories like LDAP (Lightweight Directory Access Protocol) could be used. The act of identifying responsible people for a generic human role is called *people resolution*. The user is required to perform some action. A work item is added to the task list of the user in order to inform the person about the possibility and necessity to perform specific actions on the task. A user decides to work on an activity by claiming it and unique access to the activity is provided to the user. Beside tasks also simple communication steps named *notifications* are possible where information is just transmitted to an interested party. A *notification* just informs a user in contrast to a task that holds up a process until its completion [III⁺05].

People Initiating Processes

In many cases, only certain persons are allowed to be the initiator of a certain process. A definition must be possible where the people are defined that have the right to start a process [III⁺05].

People Managing Long-Running Processes

Long-running processes might also require human interaction. In current BPEL, a timeout of a business process would mean that all the parts of the process had to be undone. Better solutions might be possible by informing a user about the timeout. The user might still be able to complete the process successfully. Again, only certain users should be able to access the process and with people assignments a business administrator of a process should be set [III+05].

Transition between Human and Automatic Services

Sometimes services can be performed with or without human interaction. An example could be a translation service. Human tasks should be supported but not necessarily be required in this case. The transition between the two ways has to be non-disruptive [III+05].

Advanced Interaction Patterns

But also more complex patterns have to be supported and handled by BPEL4People. “Separation of duties” after the *4-eyes principle* has to be addressed where a decision is made independently by two or more persons. Also excluding people from performing a special activity has to be enabled. People may already have performed the activity or people may not be authorized [III+05].

Escalations happen when tasks are not progressing as expected and modeled time constraints are not met. In this case, a notification is sent by e-mail, instant message or SMS to people that were specified as escalation recipients with a people assignment definition. Also a chain of escalations can be defined. Different escalation recipients could be informed depending on the number of escalations, for example. Escalation information must be kept but the escalation chain has to be terminated immediately once the task completed [III+05].

Also *Nominations* are necessary meaning that a supervisor nominates the ownership of a task to a special colleague in order to pass responsibility [III+05].

Chained Execution means that a sequence of process steps has to be performed by the same person. Usually, after completing a step, a user has to go back to the task list in order to see new todos. It would be inconvenient if a user always had to go back to the list after each step of a long chain. This is why a wizard-style interaction with options like “complete and claim next task” should be supported [III+05].

Features

For the link between people activities and organizational directories special people links are necessary. People links represent “the group of people that

are associated with a people activity” [III⁺05]. Corresponding to the 4-eyes principle, the owner of an approval activity is passed as a parameter to the people link of the next approval activity in order to exclude this owner. `People queries` are used to determine the actual set of people that are involved in the process. LDAP filters, SQL (Structured Query Language) queries, Java methods or an XQuery (XML Query Language) might be used for that purpose. Also priorities that may depend on some data of the process can be specified for `people activities`. Activities with higher priority might concern tasks for “gold costumers”, for example [III⁺05].

Also tasks may have priorities but priorities of tasks used by a `people activity` with a differing priority are ignored. Priorities of `People activities` override priorities of tasks. Client applications like a task inbox (with a user interface) have to present the tasks to users. User interfaces are associated with tasks. Relevant data should be presented before a task is executed. Different operations have to be executable like querying a task, claiming it, revoking the claim and completing the task. It also has to be possible to fail a task in order to indicate failures. Consequential, the lifecycle of a task can be described by different states: A task can be *ready*, *claimed*, *completed* and *failed*. Inline tasks and standalone task are differentiated. Inline tasks are a part of a people activity and are shown in constellation 1 of figure 2.7. Here, the use of the task is limited to the people activity that encompasses it. But a task can also be defined as a top-level construct of a BPEL process for enabling the use of the task in multiple `people activities` (constellation 2). In constellation 3, the task definition is done independently of any process and constellation 4 offers a Web service interface using WSDL for the task. The most generic case (constellation 5) shows a BPEL process invoking a Web service that is represented by a human task [III⁺05].

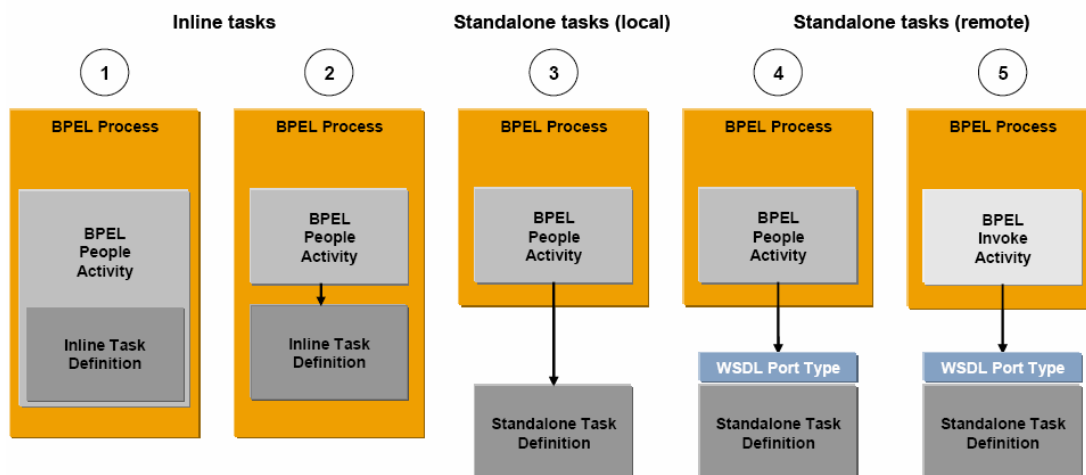


Figure 2.7: Models of Interaction between Tasks and Processes [III⁺05]

Also process context information is needed to execute arbitrary activities (including `people activities`). Sometimes a special context like geographical information is important for `people queries` in order to get fitting data [III⁺05].

BPEL4People does not define how data is rendered for user interfaces. However, it provides a default method for showing top level data of a message by using HTML forms or XForms, for example [III⁺05].

3 Strategy for Top-Down SOA Projects

3.1 Overview

For top-down SOA projects, business process models are created on a high level of abstraction. People in different positions in a company can discuss the model on this high level while technical details are still missing. A shared understanding of a business process and its purpose is needed. The process has to be understood and a detailed process analysis is essential. All existing information about the process must be collected and understood. Misunderstandings can result in an incorrect implementation of process steps and expensive changes might be necessary. Therefore, discussions of IT developers and the business unit of a company should take place in order to discuss all process steps.

Based on existing process descriptions (step 1 in figure 3.1), a business process model has to be developed (step 2) with a modeling tool. For the purpose of this thesis, extended Event-driven Process Chains were used for modeling business processes and are explained in subsection 3.2.2 (page 39). The process has to be modeled in a readable way for IT systems and help of IT personnel might be needed. The final result of the model can be subsequently translated into BPEL-code (step 3) with integrated conversion tools.

As the current supported BPEL 1.1 standard does not yet include all necessary functionalities (i.e.: no human interaction support), additional BPEL extensions might be needed for a correct further processing (step 4). Extensions have to be readable by tools of the next lower layer where the extended BPEL-code is imported. Companies developed own ways of adding the missing information in BPEL within their product stack. When different products of different companies are used, additional scripts might be necessary that offer a solution for improving the compatibility. More clarity and compatibility is aimed with the enforcement of the next BPEL standard (BPEL 2.0).

The extended BPEL-code (step 5) has to be imported in process development software (step 6) where an executable process can be built (step 7). Necessary Web services have to be available and Web service calls have to be checked for correctness. The executable process has to be deployed on a server (step 8) and it can be accessed over its Web service interface. After the implementation of a fully working project, a special graphical user interface can be developed (step 9) for a user friendly control over the process.

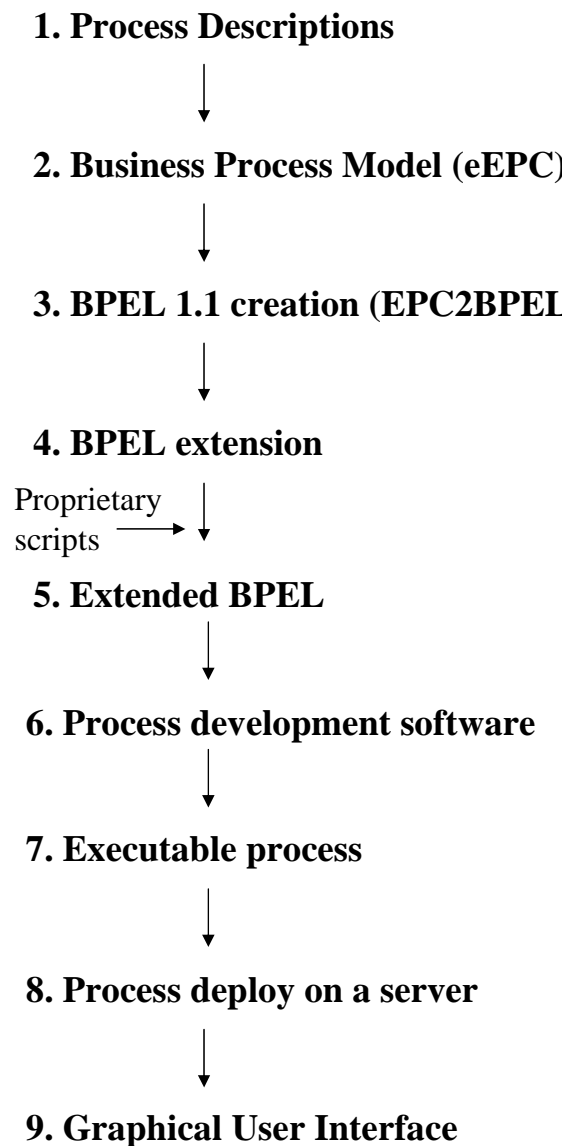


Figure 3.1: Generic proceeding (top-down)

An overview picture related to the experimental sample in this thesis is shown in chapter 4 on page 51. The actual proceeding is detailed and used software is listed.

3.2 Business View

The following pages explain the steps and methods and rules for creating a convertible business process model.

3.2.1 The Business Process Model

When creating a process model in a Service-Oriented Architecture, several aspects should be considered. The model should be concise and it should correctly represent the process flow. It should be easy to read and easily understood. As an additional requirement, the model has to be convertible into a format that can be interpreted on the next lower layer of the architecture. Therefore, the process model must be created in a language that is capable for transformation. Special model design rules represent another restriction for the model as the model has to be designed in a convertible way. For a concise modeling, Event-driven Process Chains turned out to be very convenient. Another essential advantage of EPCs is the fact that it can be converted into BPEL (EPC2BPEL).

3.2.2 Event-driven Process Chains (EPCs)

Original EPCs consist of events, functions and logical connectors and are used for the description of logical sequences of business processes. Events trigger functions which produce new events and therefore an alternating sequence of events and functions develops [Sch00] [Sei02]. As also other information beside the control flow can be relevant for business process models, the EPC modeling language was extended by the usage of elements for resources, data, time and probabilities building the *extended Event-driven Process Chain* (eEPC) [vHOS05] [LA98]. This way, also roles carrying out functions can be modeled as the people and organizational units that are responsible for special tasks can be included in the model. Data flow including input objects and output products can be described as well [LA98].

Extended EPCs represent the main modeling language for process modeling in *ARIS* (Architecture of Integrated Information Systems) by IDS Scheer AG and the language is also intensively used in the enterprise system *SAP R/3*. Static objects like organization units, data objects, application systems etc. are combined in an eEPC building a dynamic model where sequences of process steps are represented [DB07].

Processes modeled by usage of eEPCs have advantages referring to textual based process descriptions. In a company, processes can be understood, managed and changed easier when clearly arranged process models are available. When working with eEPCs, symbols for functions and events are used to describe process steps:



Figure 3.2: EPC: Event symbol [DB07]

An event (figure 3.2) symbolizes the arrival of a special condition, a defined state invoking a series of activities. System states that don't have an immediate effect on the system do not belong to the event-related states. Within the scope of information modeling, an event represents an arrived state of one or a group of information objects. Therefore, it is a passive component of the information system without decision-making authority [KNS92].

Fundamental characteristics of events within the scope of information modeling [KNS92]:

- Events can invoke functions
- Events are invoked by functions
- Events represent an arrived business state
- Events serve the specification of business conditions
- Events can refer to information objects of the data model

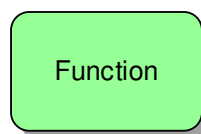


Figure 3.3: EPC: Function symbol [DB07]

Functions (figure 3.3) can be understood as tasks in an eEPC. They represent physical or intellectual activities that shall be accomplished. When designing process models, the focus is on the goals that have to be achieved rather than the way they are accomplished. On the business layer, a function represents a business activity and an active component within the information system. A function refers to what is to do rather than how to do it. It transforms input data to output data by reading, changing, deleting or creating objects. A decision-making authority concerning the succeeding

functions is carried by a function. Functions can be split up until a state is reached where further divisions would not make sense anymore for the business view. Functions are different if they differ semantically or in the number of input and output parameters [KNS92].

Process models support a more dynamic view within information models than the more static data and function models. Functions are initiated by events and the function flow related context is presented. An event that initiates a function corresponds to the fact that values of attributes are specified. By abstraction of concrete values *event types* and *function types* can be defined. An event type represents a unique defined collection of events that belong to one class because of the existence of the same attributes with specified values. The attributes carry the needed information [KNS92]. Accordingly, a function type corresponds to a unique defined collection of functions that represent tasks.

The tables 3.1 and 3.2 demonstrate the difference between abstraction and specification layer.

	Event type	→	Function type	→	Event type
Abstraction layer	Service requirement available		Create purchase requisition		Purchase requisition created

Table 3.1: Abstraction layer [KNS92]

	Event	→	Function	→	Event
Specification layer	Consulting for SOA security required		Create purchase requisition for security consulting		Purchase requisition for security consulting created

Table 3.2: Specification layer [KNS92]

A relation exists between event types and information objects as an event type can be assigned to one or more information objects. An information object can be in relationship with one or more event types [KNS92]. For finding event types from scratch, significant practical events have to be identified. If already a data model exists with complete attribute information, potential event types can be found easier. Event types can be identified by analyzing possible values of the defined attributes [KNS92].



Figure 3.4: EPC: Organisation unit symbol [DB07]

The symbol for organisation units (figure 3.4) represents staff members and can be used in eEPCs to demonstrate human interaction in a process model.

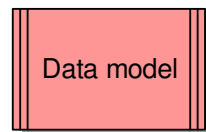


Figure 3.5: EPC: Data model symbol [DB07]

Any kind of information carrier like a document can be described by the data model symbol (figure 3.5). Data models are used as in- and output for functions.

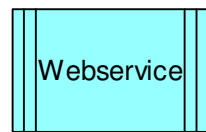





Figure 3.6: EPC: Web service symbol [DB07]

There is also a special symbol that is used to describe Web services (figure 3.6). Web services can be connected to functions in order to demonstrate that the function is executed by a Web service. Still there are other symbols that can be used when creating eEPCs. But initially, the shown symbols were tested for getting a process model that can be converted to BPEL.

Various variants of connections between event types and function types are possible by specifying *logical operators*. Figure 3.7 shows the logical operators for creating a disjunctive connection (*xor*), a conjunctive connection (*and*) and an adjunctive connection (*or*). Their function is explained in figure 3.8.



Figure 3.7: EPC: XOR, AND, OR [DB07]

Operator	Following a Function (single input, multiple outputs)	Preceding a Function (multiple inputs, single output)
OR 	OR – Decision One or many possible paths will be followed as a result of the decision.	OR – Trigger Any one event, or combination of events, will trigger the Function.
XOR 	Exclusive OR – Decision One, but only one, of the possible paths will be followed.	Exclusive OR – Trigger One, but only one, of the possible events will be the trigger.
AND 	AND – Parallel Path Process flow splits into two or more parallel paths.	AND – Trigger All events must occur in order to trigger the following Function. ¹

Note 1: It may be necessary to consider a time period during which all the events must occur in order for the AND trigger to be valid.

Figure 3.8: Logical Operators [DB07]

An Event-driven Process Chain shows which events initiate which functions. It also demonstrates which functions create which events. An example can be seen in figure 3.9. As an event type that was created by a function type also represents an activator of a following function type, a continuous chain develops. An *event type connection* represents a connection of several event types with one function type whereas a *function type connection* specifies a connection of several function types with one event type [KNS92].

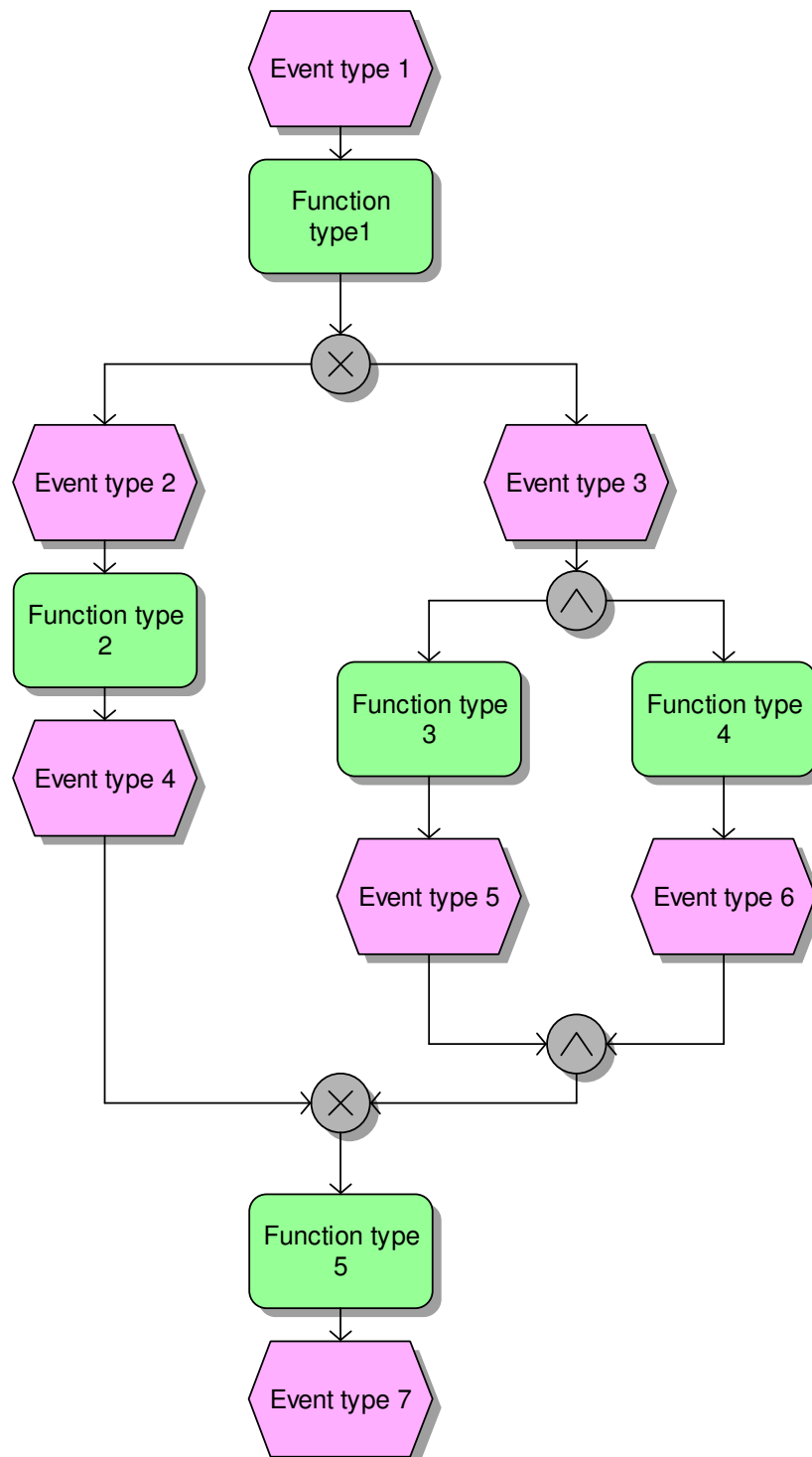


Figure 3.9: An example eEPC [KNS92]

An eEPC is suited very well for the first step of process/function modeling as well as for giving an overall view over all function and event types that belong to a special scope. It describes the chronological and logical flow of functions and it demonstrates the relations of elements of the data and function model. Therefore, it plays a central role within the scope of information modeling [KNS92].

Not only the control flow can be interesting but also the analysis of incoming and outgoing information objects of a special function type. This can be done by examining the input and output objects (attributes). For organization models with responsibility assignment and assignment of tasks, organization units can be used in an eEPC [KNS92].

3.2.3 Identifying Services or Service Candidates

A recommendable advice for process modeling is to look for existing models, ideas and functional similarities based on former projects. Existing equal sub processes should be searched and reused. The business process model should be designed in a way that reusable sub processes can develop. Sub processes might be implemented as own services. Web services executing a needed function might even exist already. Therefore, a look into the repository for reusable services is recommendable. Once a service is found, it can be integrated and referenced by the corresponding function in the model. A function is therefore represented and executed by a Web service as shown in figure 3.10.

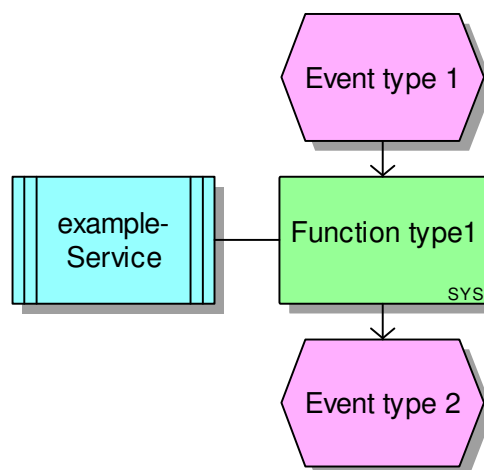


Figure 3.10: Web service call example

If the service does not exist yet, the designer can write a service description with information about the properties of the needed service. The service itself should be implemented by a developer with help of this description.

3.2.4 Designing the IT Version of the Business Process Model

It is important to create a concise business model that reflects the real process steps. However, still some modifications might be necessary to make it convertible. It has to be modified in a way that no errors happen when executing EPC2BPEL conversion and the help of an IT specialist might be needed. Rules have to be followed when designing models for later conversion into BPEL. Table 3.3 shows general rules for modeling with EPCs.

General EPC Rules
Every model must have at least one start event and one end event.
Functions and events always alternate.
Functions and events only have a single incoming and outgoing connection.
Process paths always split and combine using rules.
Multiple events triggering a function combine using a rule.
Rules cannot follow a single event.
Decisions are taken by functions.
Functions that take decisions are always followed by rules.
Rules show the valid combination of paths that follow a decision.
Events following rules indicate the actual outcomes of decisions.
Rules cannot have multiple input and multiple outputs.

Table 3.3: General EPC rules [DB07]

ARIS SOA Architect offers the possibility of validating a model before converting it. The model is checked for correctness corresponding to these rules and a validation result is shown after the check. A result page demonstrates the followed and the violated rules and gives a short description of the problem. Based on this result page, structure rules and rules for service-oriented EPCs are shown in tables 3.4 and 3.5. The result page itself can be found in appendix B.

Structure rules	Description
All functions/events have only one incoming/outgoing connection	This rule checks whether all functions and events have a maximum of one incoming or outgoing connection.
Each path must begin and end with an event	This rule checks whether all paths begin and end with an event.
No OR/XOR possible after event	This rule checks whether splitting OR or XOR rules (distributors) do not exist within a process after events.
No objects without connections may exist	This rule checks whether a model contains object occurrences without connections to other occurrences. Each object in a model must have one or more predecessors and/or successors.
Number of outgoing or incoming connections at the rule	This rule checks whether there are either exactly one incoming and a minimum of two outgoing connections at each simple rule, or a minimum of two incoming and exactly one outgoing connection.

Table 3.4: Structure rules for EPCs [AG06]

But still this information is not enough in order to be able to transform every model in a convertible way. This is why experience is needed and the designer of the IT version of the model has to be trained. After gaining lots of experience by creating many process models, the design of the first model should come closer and closer to the final IT-version of the business model in the long run.

Structure rules	Description
A business function should be carried out by one single organizational unit.	Only a business function that is connected with one single object of the organizational unit type via a relationship of the 'carries out' type is interpreted via the transformation to the BPEL process.
A system function is supported by one single object of the 'Application system type' type.	A system function may be supported by only one single object of the 'Application system type' type and is thus interpreted via the transformation to the BPEL process.
All input and output objects must be mapped to objects of the 'Class' type or be themselves objects of the 'Class' type.	All objects that are connected with functions via relationships of the 'has input' or 'has output' type should be of the 'Class' type or be directly or indirectly mapped to one or more objects of the 'Class' type to be interpreted correctly via the transformation to the BPEL process.
An 'Application system type' object supporting a system function may be connected with only one object of the 'Component' type.	According to the modeling conventions for the representation of a service in ARIS, only one single object of the 'Component' type may be connected with an 'Application system type' object via a relationship of the 'encompasses' type.
Only one single object of the 'Operation' type is connected with a system function.	Only one single object of the 'Operation' type can be connected with a system function and interpreted via the transformation to the BPEL process.
Only specific types of function symbols are used.	Only the business function, the system function and the system function target are allowed for transformation to the BPEL process.
Process contains public messaging activities.	Public messaging activities serve to specify the input and output information of the process. This information is used by other components when communicating with the process. According to the modeling conventions for service-oriented EPCs, public messaging activities can represent process steps that follow both start events and preceding end events and specify the associated input and output data objects. Alternatively, the input and output data of the process can be specified as a data object of the start or end event.
Process parallel flows, inclusive decision paths and exclusive decision paths are well-formed.	Process parallel flows should be specified by splitting and joining AND/XOR rules, or they should contain either one splitting AND/XOR rule only for which there is no other connection between their paths, or one joining AND/XOR rule only that is met by all connections.

Table 3.5: Rules for service-oriented EPCs [AG06]

3.3 IT View

In this section, the next steps are explained that have to be done for getting a runnable project. IT developers have to implement the process based on the finished business process model. The steps are explained in detail related to the practical example project in chapter 4 starting on page 51.

3.3.1 BPEL Conversion

After the final IT-version of a business process model is finished, an IT specialist gets the responsibility for the implementation of the designed process. First, the model has to be converted into BPEL. Tools usually offer BPEL 1.1 conversion where several functionalities are still missing and not all information is kept. Human interactions are not yet supported by the current BPEL-standard and the generated generic BPEL-code has to be extended by proprietary code for not losing this important information. Specific extensions are necessary that have to be compatible with solutions of the tools on the next lower level where the BPEL-code is imported. Specific scripts have to be written to be able to deal with this existing incompatibility. Hopefully, a standard solution for human interactions can be defined in the future but specific solutions are necessary until that support is guaranteed.

3.3.2 Completing the Process

BPEL-code is imported in process development software where still adoptions have to be made in order to get a deployable version of a process. The extent of required adoptions depends on the maturity and compatibility of used software. Hopefully, the necessary amount of work will be reduced once better support is guaranteed based on common standards.

An IT developer has to create a runnable version of the project. The lower the level of development the more information is needed as more details become important for the implementation. Not all details can (and should not) be modeled in a process model as a model becomes confusing once it is designed too fine-grained. For still existing obscurities and questions, a detailed communication is possible that is based on a process model that all participating parties can understand.

Services that are invoked by the main process have to be available for the implementation. They can be included if usable versions of the services exist but otherwise they still have to be implemented. Services can be implemented with process development software or existing applications can be integrated. For existing applications that do not have a usable interface yet, this interface has to be created first. A Web service interface can be created by wrapping an existing application and making it look like a Web service. This way, legacy applications can be embedded and reused in the infrastructure. It might also be necessary to include special adapters in the infrastructure for accessing existing applications. Once a service and its interface are implemented,

it can be accessed by including its WSDL-file with information about the input and output parameters, port and location of the service.

Figure 3.11 shows a general overview of usable services. The general process model is designed on the top layer. The second layer refers to the fact that atomic services or composite services can be called. Composite services again consist of atomic or composite services. Different service implementations are possible like orchestrated services with or without human tasks or applications that are wrapped using a service interface.

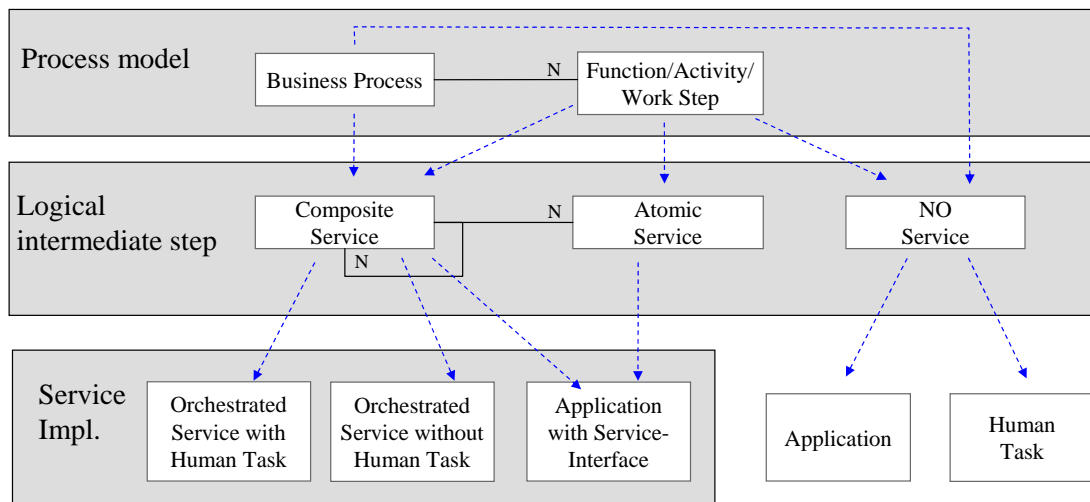


Figure 3.11: Service overview [Dai07]

4 Practical Example (SOA Live Session Project)

4.1 Motivation

Figure 4.1 shall demonstrate the complexity and heterogeneity of application architectures. A lot of time and money is spent for staying on top of things. Application integrations and exchanges are expensive and always cover a risk in such a complex environment. Service-Oriented Architectures shall help to reduce this IT complexity.

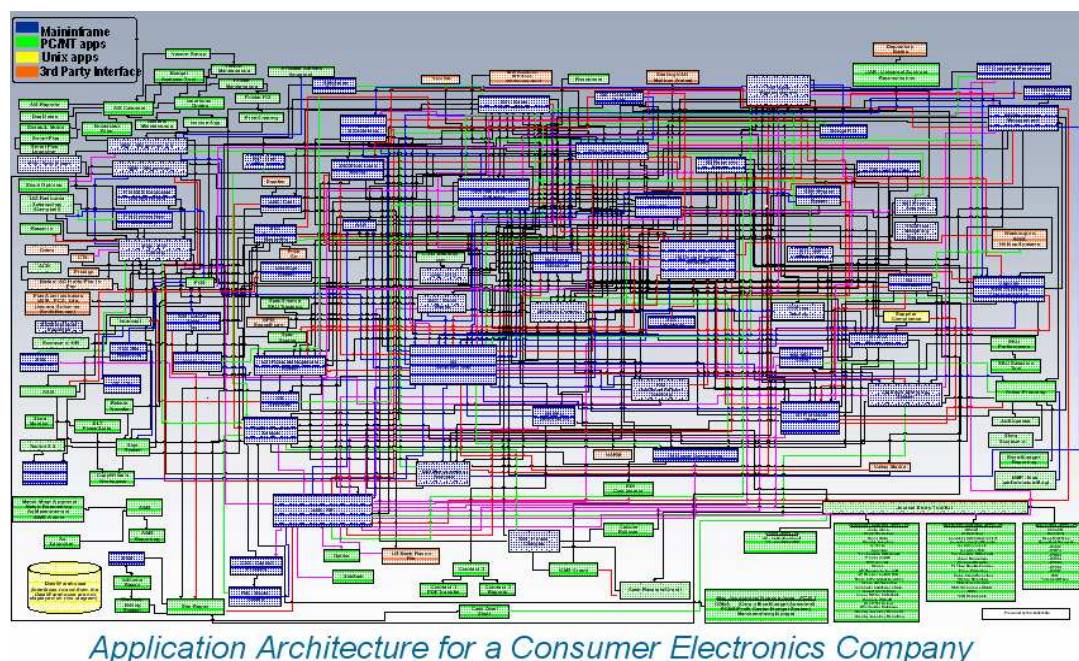


Figure 4.1: Application architecture for a Consumer Electronics Company [Spr07]

A prototype of a real process of DaimlerChrysler AG should be created with SOA design rules. This way, the current state of SOA concerning the existing tools for developing SOA projects should be analyzed.

A real existing process of the company should be implemented. However, it should be simple as the individual process steps were not the focal point. The new architecture and the top-down approach should be examined. The purchase requisition process

of non-productive material was chosen for the prototype because of its simplicity and publicity. It is an approval process that many coworkers come in contact with. Therefore, process steps of the prototype are known and can be easily understood.

Up to now, the process is implemented with the client-server application Lotus Notes of IBM and its Lotus Domino server. The control logic is implemented using Lotus script and the SAP system NACOS (New Accounting and Controlling System) is accessed for the creation of purchase requisitions. Programming is necessary for changes of the process. A service-oriented approach might replace this process one time because of its flexibility.

With the prototype, a better understanding of SOA with all its benefits and disadvantages should be gained. The prototype should demonstrate the flexibility and agility of SOAs and this is why a process step should be changed. The change should be processed with all consequences until the new process result could be shown. Based on the experiences of the project, further investment on time, effort and money for SOA projects should be discussed.

4.2 Overview

Before implementing the process, existing process description documents had to be studied (step 1 in figure 4.2). Based on this information, the process could be modeled in ARIS SOA Architect (step 2). ARIS SOA Architect is a graphical editor that belongs to the ARIS Implementation Platform and that has a facility to produce BPEL-code. However, for a correct BPEL conversion, a model has to be created in a particular way. Several rules have to be followed that were described in chapter 3.2.4 starting on page 46. This is why the model had to be modified and an IT version of the process had to be created (step 3). The finished model was converted into BPEL-code (BPEL 1.1 + WSDL) with the conversion tool EPC2BPEL of ARIS SOA Architect (step 4). Human task information was lost during the BPEL conversion as it does not belong to the supported BPEL standard. Therefore, the missing information had to be added to the code. Two proprietary scripts were necessary (step 5): The first script extended the BPEL code by the lost human task information. The second one modified the code and prepared it for an import in IBM's *WebSphere Integration Developer* (WID). The modifications were necessary because of compatibility problems between exported ARIS BPEL and IBM BPEL.

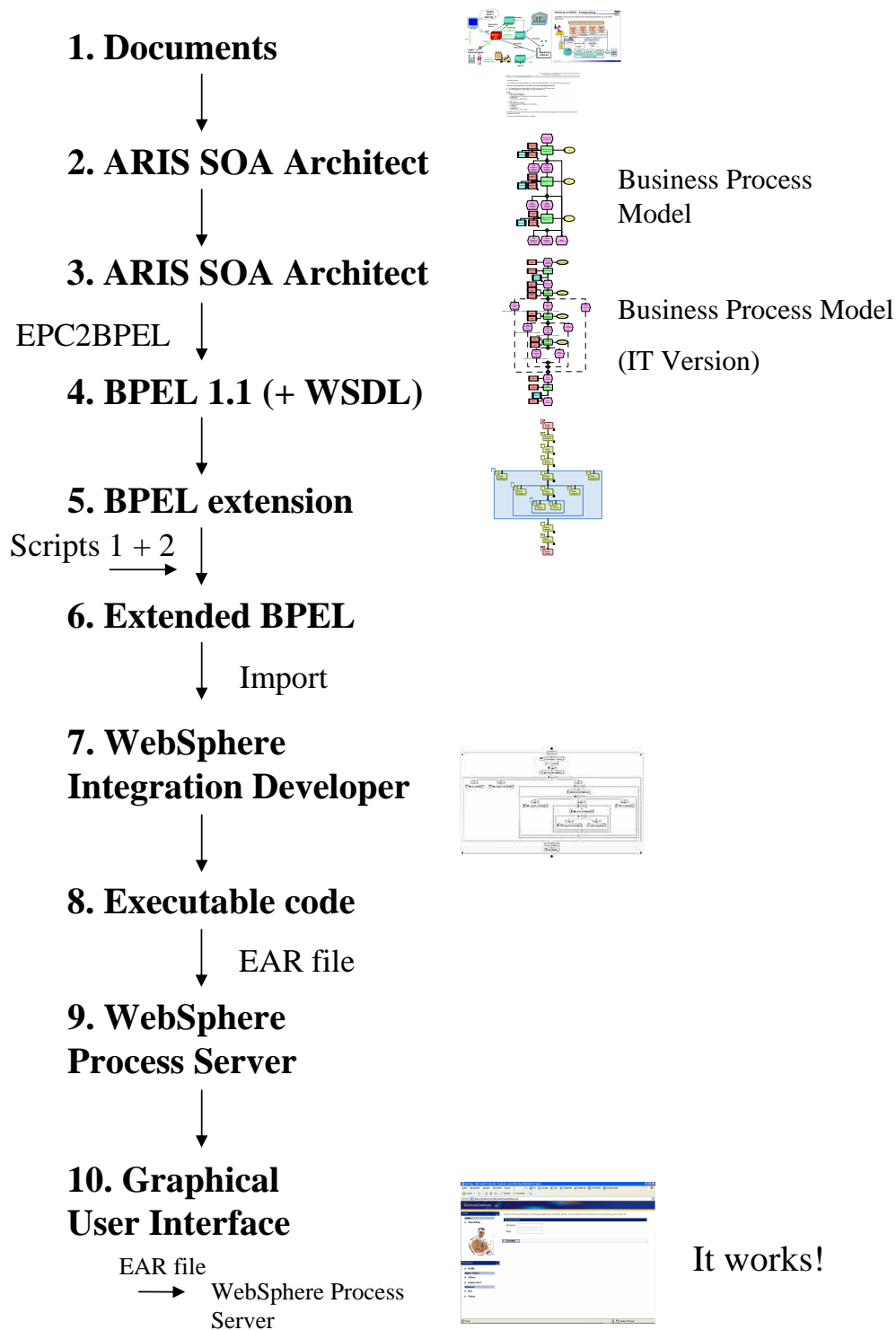


Figure 4.2: Proceeding (top-down)

After the import of the extended BPEL (step 6), the process was completed with all Web service calls in WebSphere Integration Developer (step 7). A SAP application and a SAP Web service were used that run on a *SAP Discovery System*. They are explained in subsection 4.4.2 starting on page 71.

With WebSphere Integration Developer, an executable project could be implemented (step 8). An EAR file (Enterprise Application Archive) was created that included the project according to the J2EE (Java 2 Enterprise Edition) standard. The EAR file was deployed on the (integrated) Process Server (step 9) and a working project was implemented. The WebSphere Process Server is mounted on top of the WebSphere Application Server. It offers a process container for deploying processes and it extends the WebSphere Enterprise Service Bus.

Also a graphical user interface with the corporate design of DaimlerChrysler AG was created for the process (step 10). WID allows the generation of a GUI that is based on JavaServer Faces (embedded in *JavaServer Pages* (JSP)). The GUI was changed afterwards to represent the corporate design and to fulfill the needed functionality. Again, an EAR file was created for the GUI and deployed on the Process Server.

The GUI was created for the purpose of demonstrating the process in a DaimlerChrysler executive management presentation. During the presentation, a change of the process model was shown. The model was modified live in ARIS SOA Architect. Again, BPEL-code was generated and imported into WebSphere Integration Developer where the necessary adjustments were made. The project was deployed again on the Process Server and the result demonstrated the possibility of fast changing processes in a Service-Oriented Architecture. The implemented change request is explained in chapter 5 starting on page 87.

4.3 Business View

Corresponding to the top-down approach, a process has to be modeled before it can be implemented. The following pages address the tasks of a business unit member.

4.3.1 Assessment

When creating a process model, the process itself has to be understood first. Information about the process has to be collected. Existing sheets, documents, tables or other files might be available providing information about the process. In this specific example, an animated *PowerPoint* slide helped to get an insight in the process (figure 4.3). Also a PDF and an e-mail description were available and can be found in appendix A. However, too much implicit technical information was needed for a complete understanding of the process description and still further information was necessary. Meetings with the business unit of DaimlerChrysler AG helped to get a better understanding.

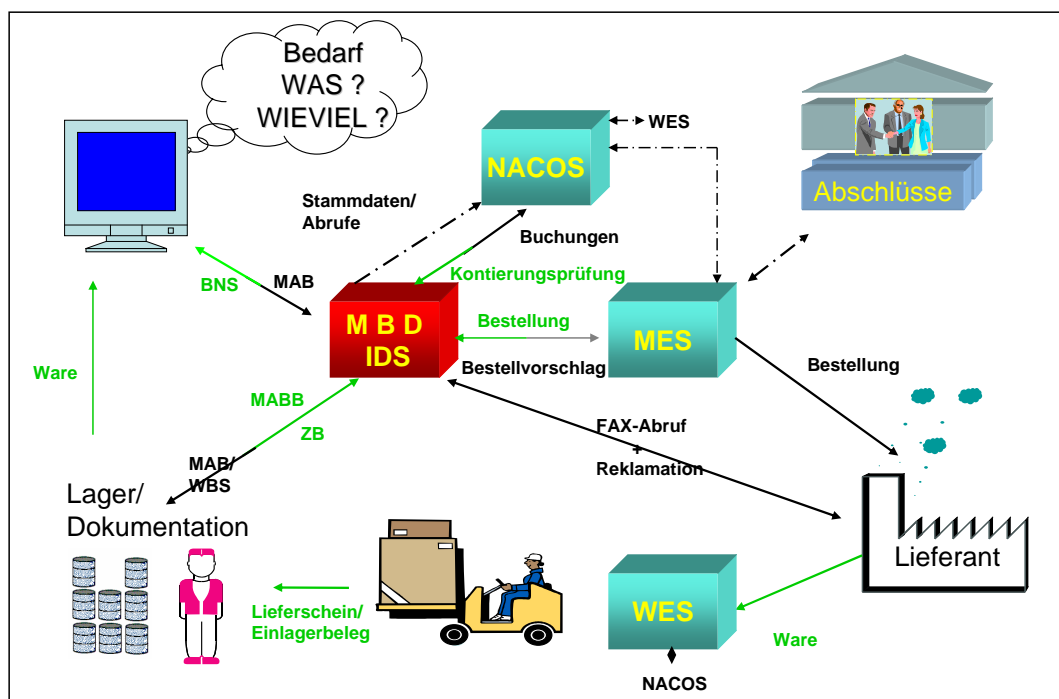


Figure 4.3: Acquisition process

Figure 4.3 shows the existing process implementation. The cuboids represent IT systems and detailed acronym descriptions for all systems can be found in appendix A. A material requisition note (*MAB*) is created in *MBD/IDS* and information about the necessary master data for the chosen material is looked up in *NACOS*. After an ac-

count assignment check, a book entry is entered in NACOS and a purchase requisition approval process is started where qualified managers approve or decline a purchase requisition. The purchase requisition approval process is not shown in picture 4.3 but an additional document provided information about it (figure 4.4). The process is implemented using the *Lotus Notes System* of IBM which communicates with NACOS. E-mails are sent to the managers who check the requisitions. If a requisition is approved, an order proposal is sent from MBD/IDS to MES ("Material-Einkauf-System") (which was lately replaced by *GLOBUS* (Global Buying System)) .

Depending on MES and general financial agreements ("Abschlüsse"), the actual order can be created and sent to the deliverer ("Lieferant"). WES ("Wareneingangs und -abgleichsystem") is informed about the order and communicates again with NACOS for goods adjustment. After the ordered products arrived in stock, MBD/IDS are informed.

In this specific example, the description of the whole process was very coarse-grained but the process became clearer after further discussions with the business unit of DaimlerChrysler AG. The purchase requisition approval part was chosen to be implemented for the prototype because of its simplicity and understandability. It is shown in figure 4.4. Depending on the amount of a purchase requisition (i.e.: 150.000 EURO), different managers have to approve the requests. The process steps are explained in detail in subsection 4.3.2 on page 58. The human interactions that take place during the process steps enable a way of interfering and the course of the process can be shown very demonstratively.

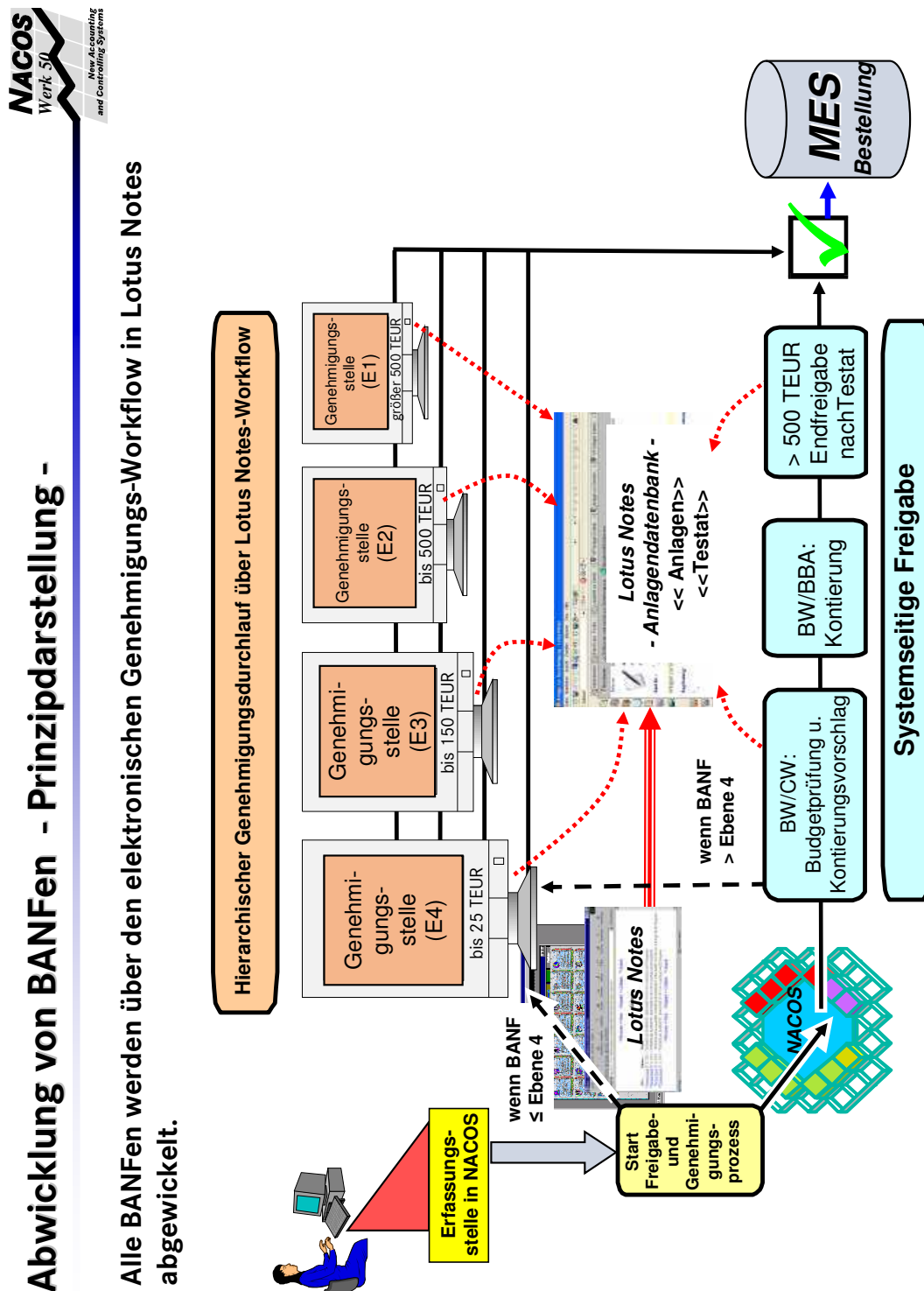


Figure 4.4: Purchase requisition process description

4.3.2 The Business Process Model

After collecting information about a process and after understanding the steps, the creation of the business model can be started. There are several modeling tools of different companies that can be used for modeling business processes. However, for this specific example, Event-driven Process Chains have been used for laying out the process workflow and a BPEL conversion should be possible. As ARIS modeling tools of the company IDS Scheer AG were already known and used for designing workflows within the company, the special version ARIS SOA Architect (Version 7.0.2.173990, Patches: 7.0.2.167238 and 7.0.2.173990) which supports eEPCs and EPC2BPEL conversion was chosen for creating the model of the process. Modeling with help of the business unit of DaimlerChrysler ended in the model shown in figure 4.5.

As this specific example is used in order to describe further steps of process modeling and implementation, a short model description is given:

After the correct creation and storage of a purchase requisition, a senior manager (2nd line manager, DaimlerChrysler AG uses the term: E2 (level 2)) is informed about the creation. The senior manager gets a PDF file with the description of the requisition. The purchase requisition that was generated in *NACOS10* has to be checked and approved or declined depending on the decision of the senior manager. For requisition prices higher than 150.000 Euro, the senior manager alone cannot decide on the approval. This is why the requisition has to be checked again by the director (3rd line manager, DaimlerChrysler AG: E2) who is authorized to approve or decline purchase requisitions smaller than or equal to 500.000 Euro. The procedure repeats also for the vice president (4th line manager, DaimlerChrysler AG: E1) who is responsible for purchase requisitions that exceed 500.000 Euro and reach up to 5.000.000 Euro.

EEPCs are characterized by good readability and can be understood pretty well depending on the effort of the modeler. Documents, tables and other files frequently differ very much and often don't follow strict design rules. Uniform models can be created based on a modeling language like eEPCs.

In the purchase requisition example, the eEPC turned out to be much more readable and understandable than PowerPoint or other document descriptions. For a complete understanding of eEPCs, still implicit technical know-how is needed, but to a much smaller degree than in lots of other ways of modeling. This first version was modeled with help of the business unit and common conventions for modeling at DaimlerChrysler were followed for this model:

- An eEPC should never start with a function but with an entry event.
- Roles that are responsible for a special task represented by the function symbol should be positioned on the right side of functions and data objects should be arranged in a readable way: Incoming data objects should be positioned on top of outgoing ones on the left side of functions.

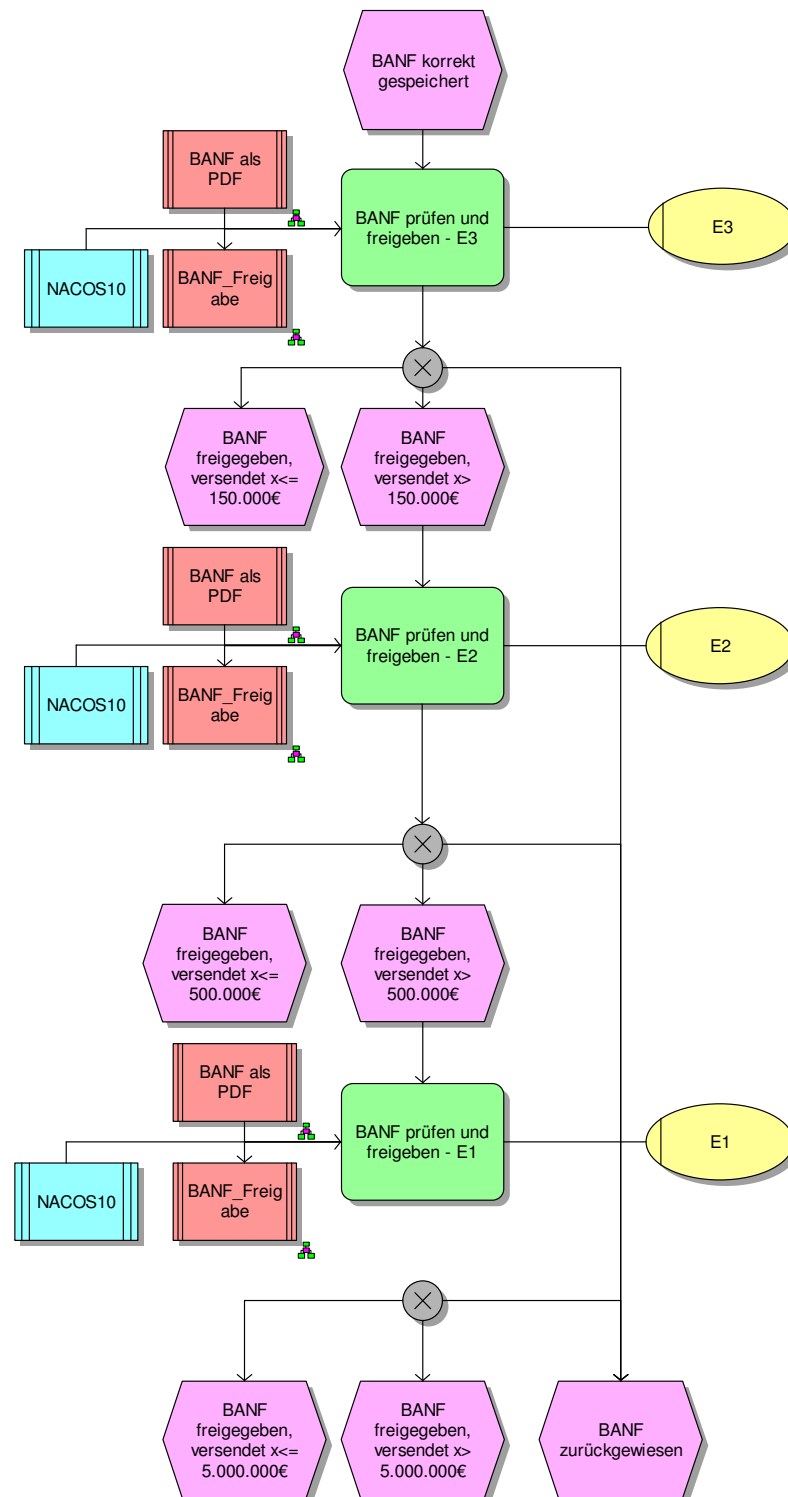


Figure 4.5: Draft eEPC



Figure 4.6: Legend for figure 4.5

- Participating application systems are positioned leftmost in the model.

This way, functions and events alternate in the middle of the model and a reader can fast follow and understand the clearly arranged process steps.

- Conditions for cases are usually mentioned in the names of events.
- An XOR-element separates the different possible cases.

The green triangular symbol next to the data objects relates to a hidden model behind the symbol that describes the symbol more detailed on an underlying layer. Varying symbols are available on the underlying layer and the objects are mapped to classes there.

4.3.3 Identifying Services or Service Candidates

Business process models have to be changed and extended very often until a representative model is created for the process. Services can be added or dropped and further process steps might be included in the model. For the example project, additional requirements for the project were defined and the business model grew. Two services should be added to the business model. One service should be responsible for the creation of a real purchase requisition object on a SAP Discovery System. The other service should perform an update on that created purchase requisition. No service repository was available but as the two services did not exist yet anyway in a usable way, they had to be developed. The services were implemented with help of WebSphere Integration Developer (WID 6.02, interimfix 003, build id: 6.0.2ifix003-20070123_1524) and are explained in chapter 4.4.2 starting on page 71.

4.3.4 Designing the IT Version of the Business Process Model

Depending on the experience, models created in cooperation with the business unit are usually not yet ready for an IT implementation. Also in the example process, first attempts of converting the designed eEPC to BPEL-code and importing it into WebSphere Integration Developer aroused many problems based on design failures and incompatibility. This is why the conversion rules of ARIS SOA Architect had to be understood and followed that were explained in chapter 3.2.4 starting on page 46. Several meetings with employees of both companies, IDS Scheer AG and IBM Deutschland GmbH were necessary to find a model that both passes the evaluation check and can be imported in WebSphere Integration Developer the best way possible. The result of the adjusted process contains the two services for creating and updating a real purchase requisition and can be seen in figure 4.7.

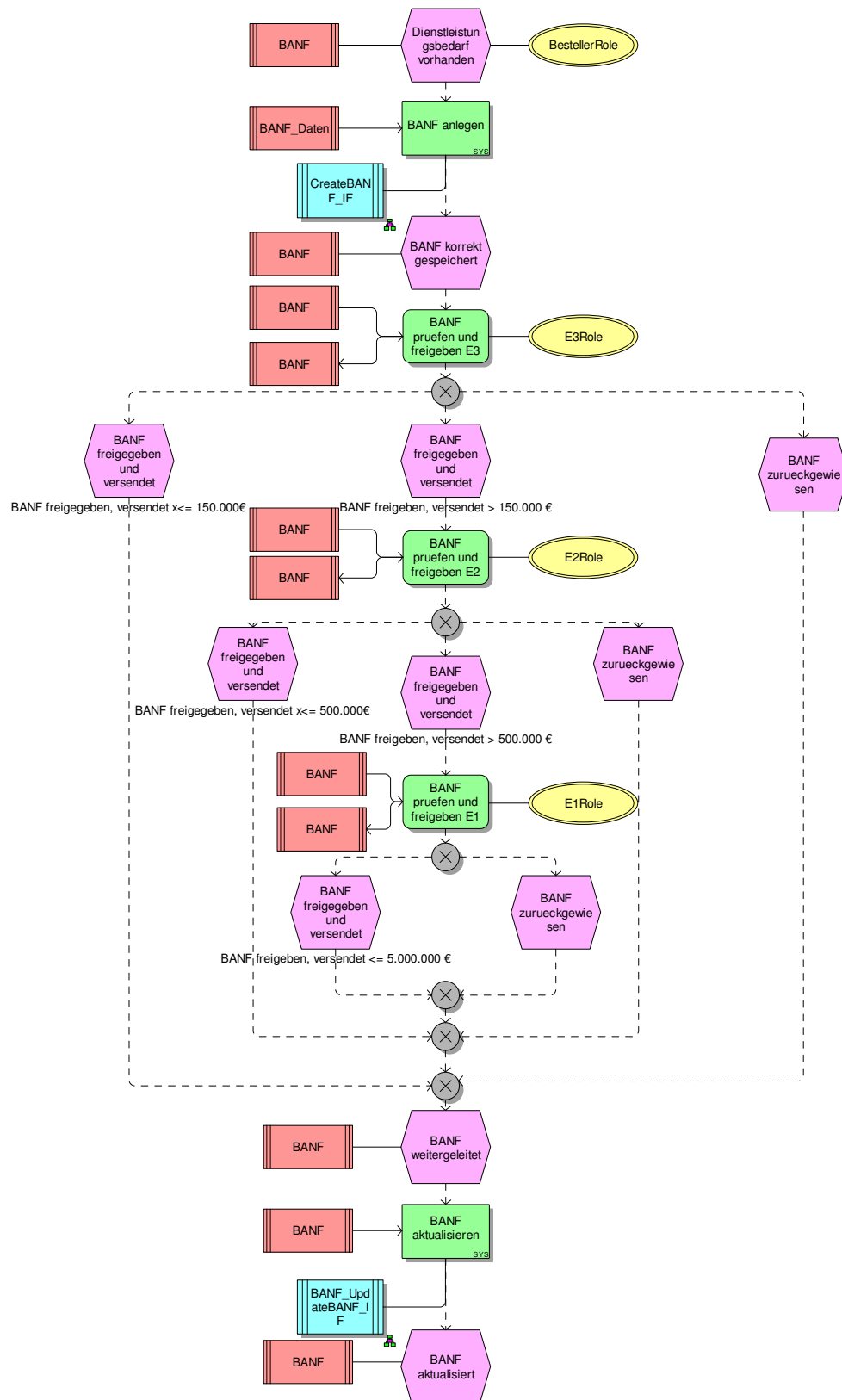


Figure 4.7: The final eEPC of the business process

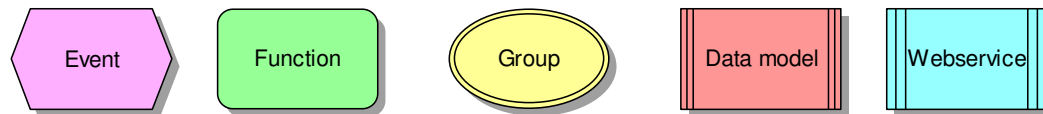


Figure 4.8: Legend for figure 4.7

The enhanced process model for the example process starts with a service requirement of an orderer. The orderer must belong to the group “BestellerRole” and is able to create a new purchase requisition. For the prototype, a manager (1st line manager, DaimlerChrysler AG: E4) creates it. A real purchase requisition is generated by calling a Web service on a SAP Discovery System. All the used Web services in the model were implemented in WebSphere Integration Developer beforehand and the WSDL-files were imported in ARIS SOA Architect. This way, the Web services were selectable and callable within the process. After the creation of the purchase requisition, the approval process steps start:

A senior manager (2nd line manager, DaimlerChrysler AG: E3) gets informed about the requisition and can decide if it is approved or declined. In case of approval and for requisition amounts of greater than 150.000 Euro, a director (3rd line manager, DaimlerChrysler AG: E2) gets informed. The Vice president (4th line manager, DaimlerChrysler AG: E1) is responsible for purchase requisitions that exceed 500.000 Euro and are smaller than 5.000.000 Euro. Even higher requisitions are declined automatically in this process model.

After the approval steps, another Web service is started on the SAP Discovery System where the created purchase requisition is updated. The update service actualizes the purchase requisition with information about amount, approvers and the final result. Figure 4.9 demonstrates the whole business process.

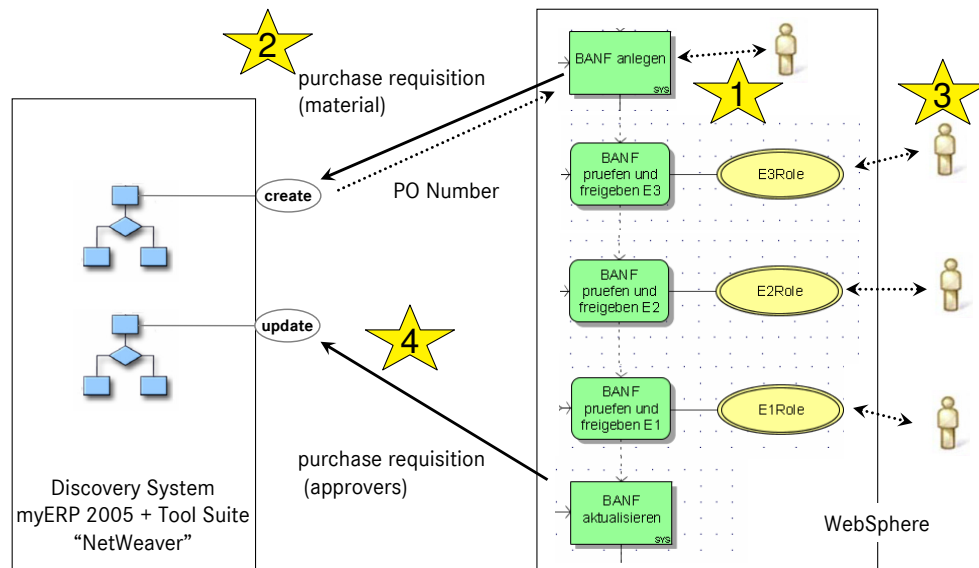


Figure 4.9: Business process

Data objects have an *Unified Modeling Language* (UML) representation with all attributes in ARIS SOA Architect. The UML representation of the *XML Schema Definition* (XSD) of the used data object "BANF" is shown in figure 4.10.

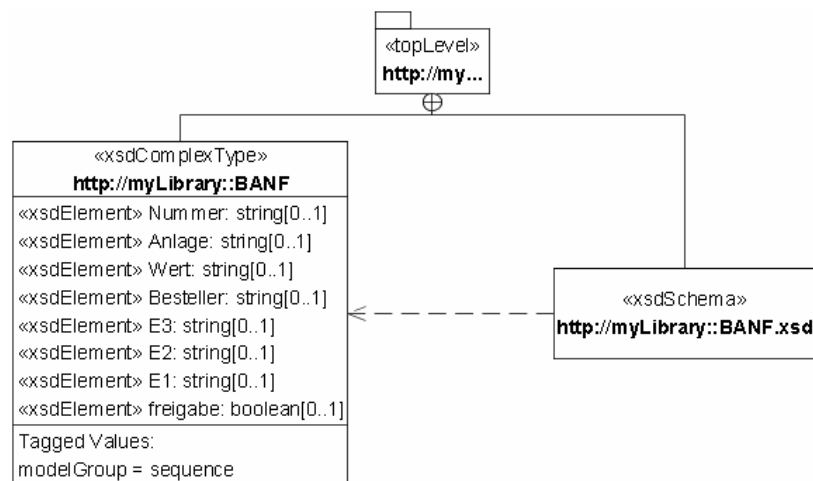


Figure 4.10: UML representation of the data object "BANF"

4.4 IT View

The following pages address the tasks of an IT developer after getting the business process model. Again, the tasks are related to the practical purchase requisition example.

4.4.1 BPEL Conversion

With the IT version of a process model, the IT developer can generate the necessary BPEL-code that is needed for the process implementation. Although the shown model of the example process (figure 4.7 on page 62) passes the validation check, the BPEL code still does not meet all requirements for a correct import into IBM's WebSphere Integration Developer.

Before the conversion, still the needed logic for the different possible paths that can be taken has to be defined in a readable way. Condition expressions have to be defined for the branches in the eEPC. This information can be included as attribute information for each branch and looks like the following:

Listing 4.1: Condition expression for branches

```
1 bpws:getVariableData("BANF", "/freigabe") = true() and  
2 bpws:getVariableData("BANF", "/Wert") <= 150000.
```

As ARIS SOA Architect supports BPEL 1.1 where human interactions are not specified, human task information is lost during the conversion. As the process contains several human tasks, there was a need for finding a general solution for exports with human tasks. The human task information had to be added to the BPEL-code and IDS Scheer AG wrote proprietary scripts for keeping this information. The red arrows in figure 4.11 show the way that has to be taken to create compatibility between ARIS SOA Architect and WebSphere Integration Developer. With ARIS BPEL vendor extensions, human task information is kept. WebSphere Integration Developer already implements functionality that is included in WS-BPEL 2.0 [Kra06] and human tasks are implemented according to BPEL4People that was introduced in chapter 2.8.3 (page 33).

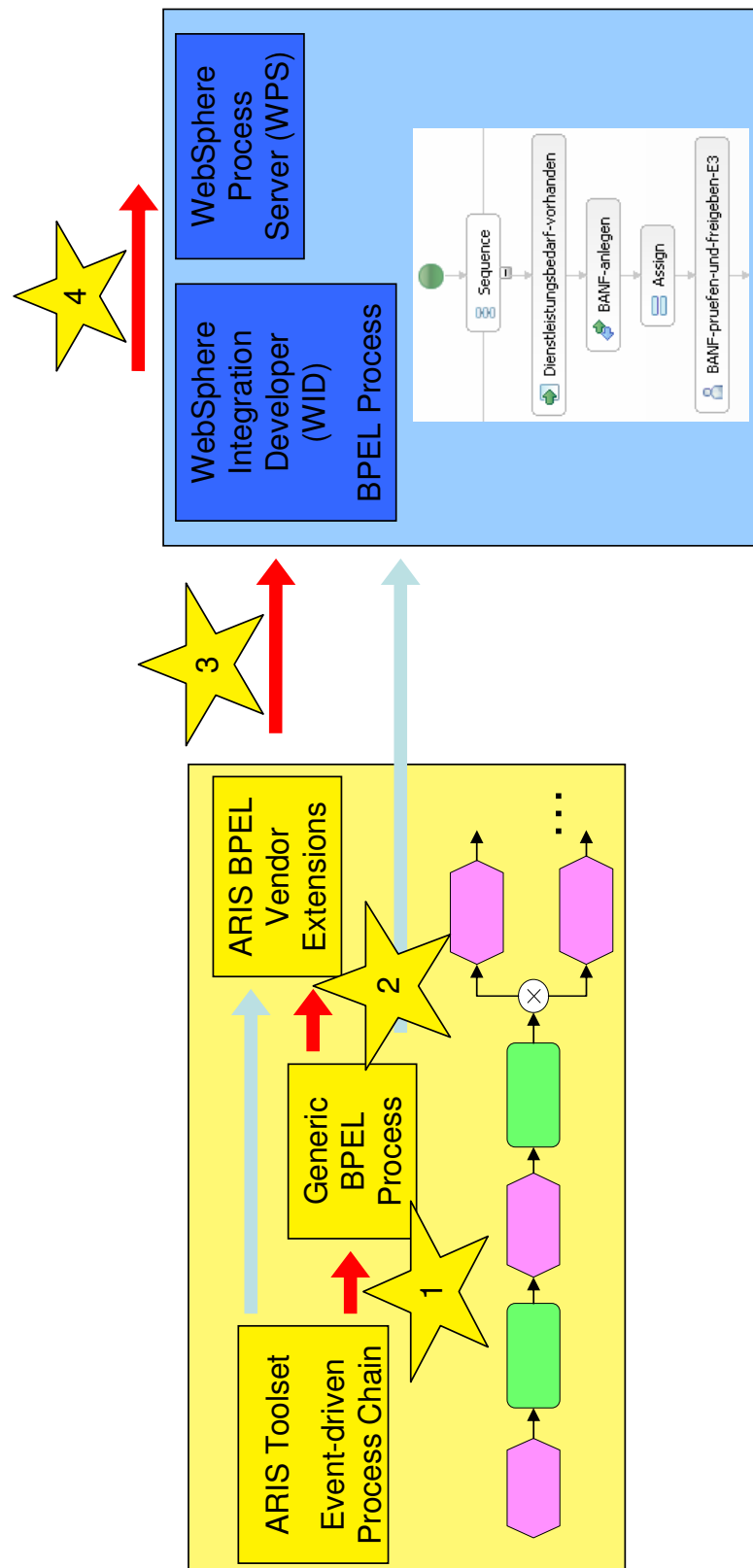


Figure 4.11: Development procedure

The first proprietary script of IDS Scheer AG accomplishes the following tasks for keeping human task information:

The start event (“BANF-fachlich-freigeben”) of the eEPC is determined. A partner link named “null” and a *wsdlPortType* (“HumanTaskPT”) is created and connected to it in the subagent BPEL allocation diagram (figure 4.12). The script also takes care of filling attributes of the start event with required namespace information.

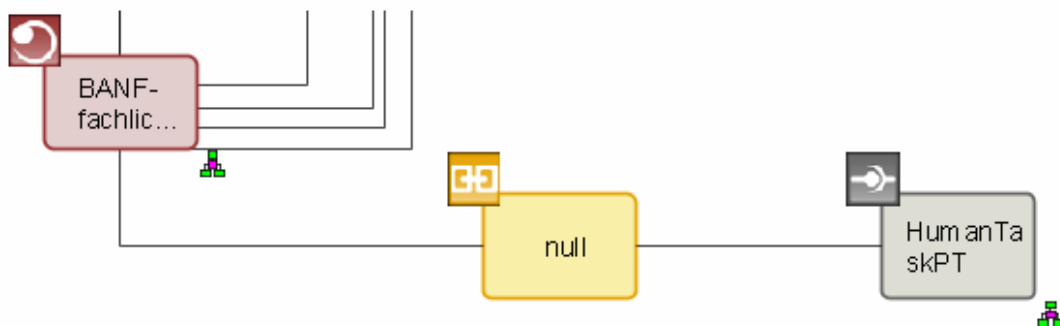


Figure 4.12: Human task extension in the BPEL allocation diagram

For every BPEL-extension (human tasks), the name of the corresponding organization unit in the eEPC is taken and inserted as name attribute. All the subsequent BPEL allocation diagrams are edited by adding partner links with the name “null” and operations (“carryBANF-pruefen-und-freigeben...”) as can be seen in figure 4.13.

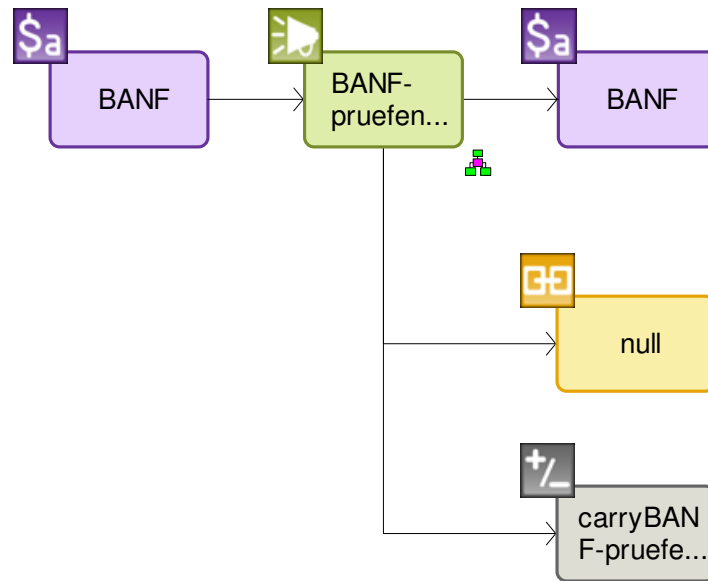


Figure 4.13: A completed BPEL allocation diagram

The name of operations is combined with the names of corresponding organization units (i.e.: “carryBANF-pruefen-und-freigeben-E2”), input and output variables are built and the port type is specified (figure 4.14).

«interface»
«wsdlPortType»
http://ids-scheer.com::HumanTaskPT
«wsdlOperation» carryBANF-pruefen-und-freigeben-E2(in input: BANFMT, out output: BANFMT)
«wsdlOperation» carryBANF-pruefen-und-freigeben-E1(in input: BANFMT, out output: BANFMT)
«wsdlOperation» carryBANF-pruefen-und-freigeben-E3(in input: BANFMT, out output: BANFMT)

Figure 4.14: Generated port type with operations

The graphical representation of the BPEL-code on the top layer can be seen in figure 4.15. The invoke symbol that before just represented Web service calls (create/update a purchase requisition) was also chosen to represent human interactions (also implemented in script 1).

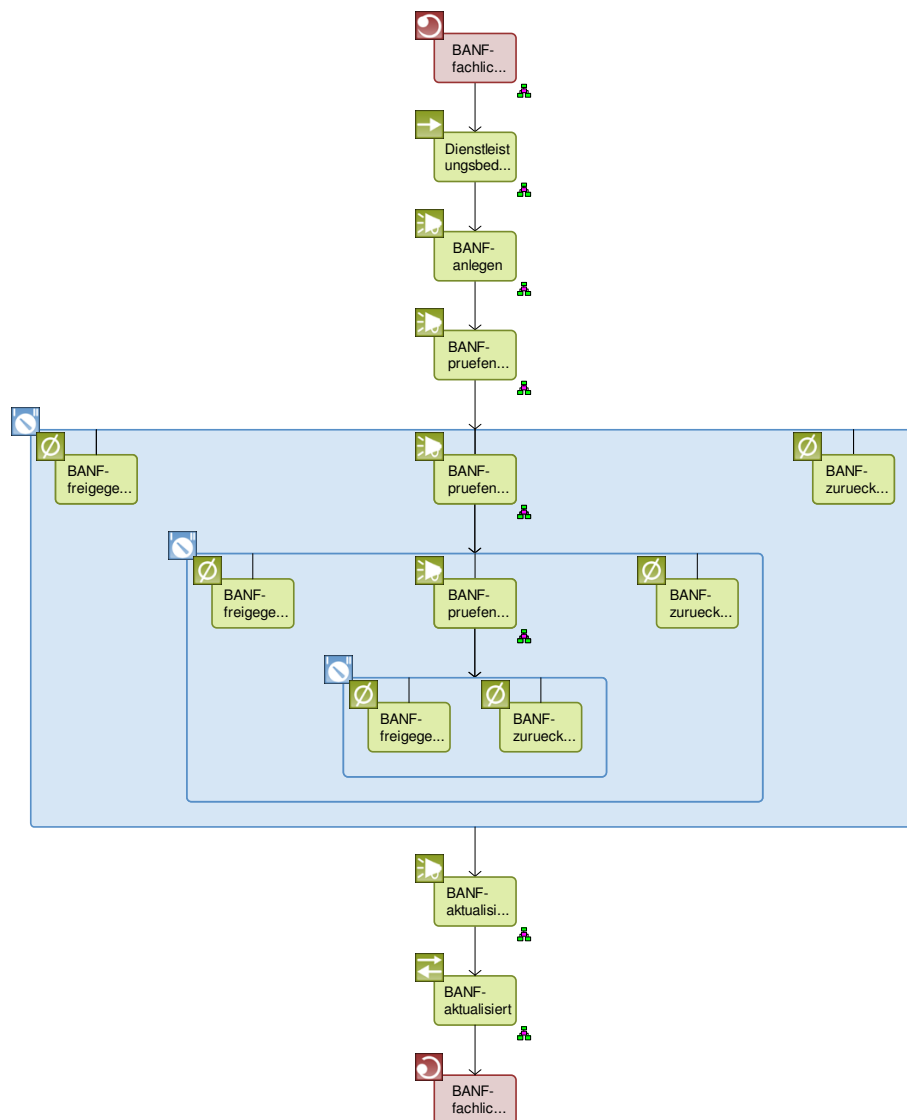


Figure 4.15: Graphical demonstration of the generated BPEL-code in ARIS SOA Architect

As the BPEL-export did not contain authorization information, also authorization rules had to be addressed for the different human tasks. After discussions with IBM Deutschland GmbH and IDS Scheer AG, also this information could be kept by editing the first script. Even special symbols that represent groups, users or other organizational units could be determined (figure 4.16).

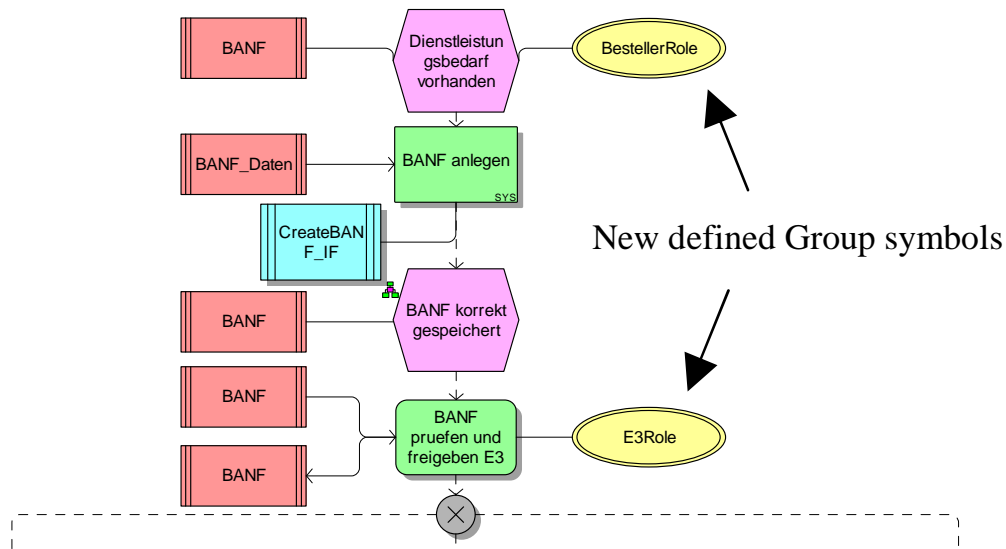


Figure 4.16: Group symbols

The script extends the BPEL-code by information similar to the following:

Listing 4.2: Human task extension in BPEL-code

```

1 <wpc:staff>
2   <wpc:potentialOwner>
3     <sss:verb>
4       <sss:name>Group Members</sss:name>
5       <sss:id>Users</sss:id>
6       <sss:parameter id="GroupName"><![CDATA[E3Role]]></sss:parameter>
7       <sss:parameter id="IncludeSubgroups">>false</sss:parameter>
8     </sss:verb>
9   </wpc:potentialOwner>
10 </wpc:staff>

```

Managers have the right to approve and decline special purchase requisitions and this way, authorization information is written in a syntax that can be understood successfully by WebSphere Integration Developer.

After exporting the code, a second script gets necessary human task information from a generated WSDL-file ("nullLT.wsdl"), combines this information with the generated BPEL-file ("BANF-fachlich-freigeben.bpel") and makes further smaller adjustments to enable a successful import into WebSphere Integration Developer on the next lower level.

4.4.2 Completing the Process

The BPEL-code reflects the designed business process model. An IT developer still has to work with the project after importing the code in a tool on the next lower level. Ideally, only small changes will have to be made. However, depending on the maturity level of the tools and the quality of the business model, still several adjustments might be necessary.

Also in the example process still some work had to be done in order to make the process deployable on the integrated Process Server after importing the exported BPEL-code from ARIS SOA Architect. Figure 4.17 demonstrates the three kinds of services that had to be worked on.

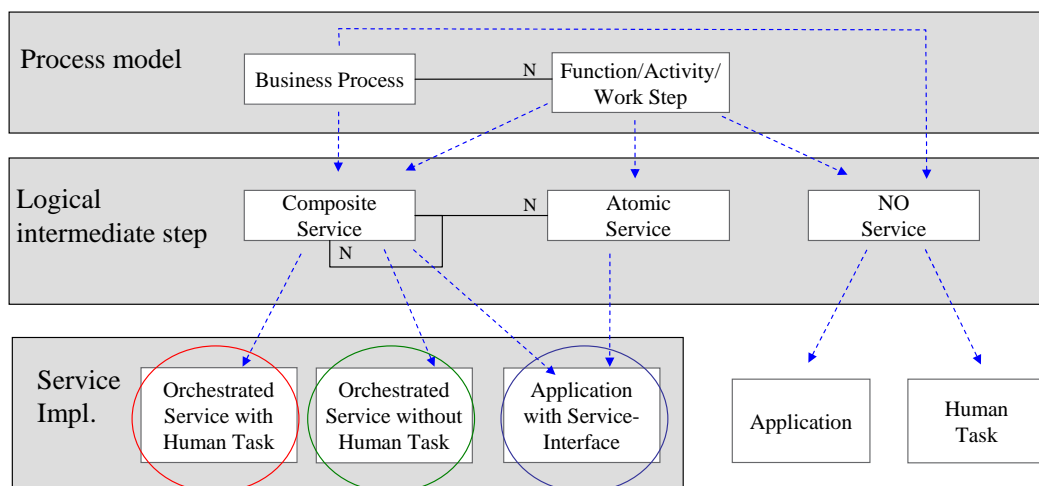


Figure 4.17: Service implementation overview [Dai07]

After the import, the orchestrated service with human tasks (red circle) was already built but still a lot of adjustments had to be made in order to make it runnable:

- As the BPEL-import did not work properly concerning information about invoked Web services, a manual WSDL-file import was necessary. A library containing all Web services was built with the import of the service descriptions and it was linked to the imported project by setting dependencies.
- Instead of using message type variables, data type variables were chosen for all input and output variables. Beforehand, message type variables were selected automatically after the import.

IBM executes SOAP-calls by using the “doc/lit wrapped” style where parameters are wrapped adding another XML element with the name of the operation.

This way, message type variables with wrapped content are generated. However, using message type variables requires assignments for setting and getting the content of variables. Instead of creating additional assignments, nested data type variables should be directly used. The implemented feature *message (un)bundling* adds a wrapper element automatically for service calls and removes it for the returned results. This mechanism enables working directly with data type variables, saves many assigns and variable definitions and was used for the process. Therefore, the right data type variables with their corresponding reference partners and operations had to be selected for the receive, the reply, for all Web service invocations and human tasks.

- An extra assignment had to be inserted after the invocation of the Web service which creates a purchase requisition (see figure 4.19). The assignment was necessary for assigning the purchase requisition number to the number field of the used business object “BANF” (figure 4.18). The number had to be assigned in order to be available for the update service in the end of the process where the correct purchase requisition has to be updated.

BANF	
Nummer	string
Anlage	string
Wert	string
Besteller	string
E3	string
E2	string
E1	string
freigabe	boolean

Figure 4.18: Used business object

- Only human interactions that were connected to process symbols in the ARIS model could be exported and imported correctly. However the support for human tasks connected to events was not yet covered and was therefore not overtaken in WebSphere Integration Developer. A human task had to be created manually for the starter of the whole process. Also an administrator for the whole process had to be set. If no administrator is specified, the initiator of the process gets administrative rights for all process steps by default.

After all last adjustments, the process was deployable on the integrated Process Server and could be tested with IBM's *BPCExplorer* (Business Process Choreographer Explorer). Figure 4.19 and 4.20 show parts of the final BPEL model. The whole model can be found in appendix A.

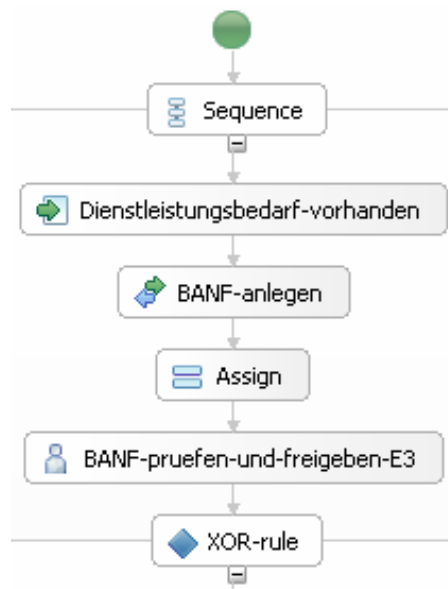


Figure 4.19: BPEL process in WID (1)



Figure 4.20: BPEL process in WID (2)

Also the Web services that were used in the business process model had to be available. Figure 4.21 shows the implemented business process that runs on the integrated Process Server. Also the Graphical User Interface is offered locally on the Process

Server. It communicates with the business process and is explained in section 4.5 starting on page 80. An external SAP application and an external SAP Web service that both run on a SAP Discovery System are invoked by the business process.

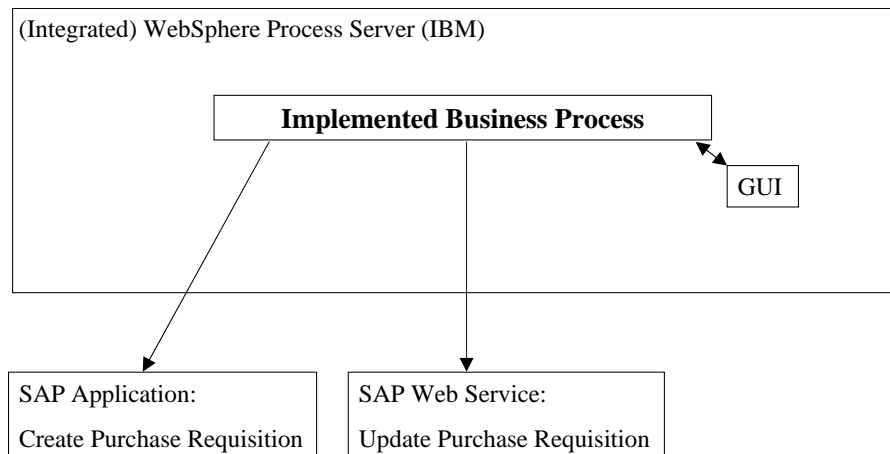


Figure 4.21: Service calls

WebSphere Integration Developer was used to make necessary adjustments to access the remote located components. As the interfaces were complex, extra Web services were created in order to call the components. The Web services were deployed locally on the Process Server and are shown in figure 4.22. They implement interface mappings and could be integrated easily in the business process. One Web service uses the adapter *SAP Java Connector* (SAP JCo) for the application invocation. A *Business Application Programming Interface* (BAPI) of SAP AG could be accessed by using the adapter. The second one calls an application using a Web service interface. The Web service interface was created by wrapping the existing interface. SAP created this interface and offered a WSDL for accessing the Web service.

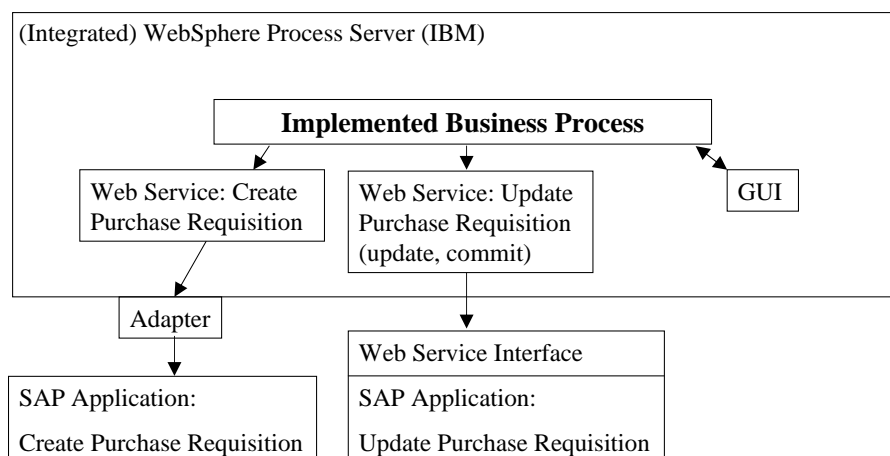


Figure 4.22: Service calls with extra services

The Service for the Purchase Requisition Creation

The service for the creation of the purchase requisition was implemented by wrapping an existing application and creating a service interface. This way of implementing services is represented by the blue circle in figure 4.17 on page 71. As the application for the purchase requisition creation was running on a SAP discovery system and had to be called from an IBM tool, an adapter was necessary. SAP Java Connector was chosen to enable this communication. The JCo packet supports both, Java to SAP system as well as SAP system to Java calls. The service should be able to create a purchase requisition and all the needed parameters had to be handed over to the application on the SAP system.

As the interface for the creation of the purchase requisition was very complex, an interface map was needed for mapping the output parameters of the process to the input parameters of the SAP system application. A Web service binding (“CreateBANF_IFExport1”) was created for the interface map (“BANF2SAPBANFIntfMap”) and the map was also connected to the EIS binding of the SAP system (“SAP_EinkaufsSystem”) as shown in the assembly diagram (figure 4.23). Assembly diagrams in WebSphere Integration Developer demonstrate which components communicate with each other.

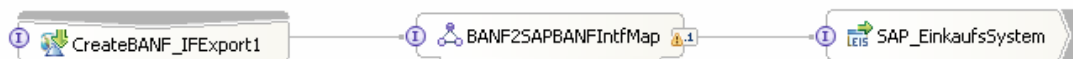


Figure 4.23: Interface map in the assembly diagram

Interface maps are necessary very often and are of high importance for enabling communication among different services with different interfaces. When focusing on the interface map of the example project, the operation “BANFRequest” was mapped to the operation “createSapBANFWrapper”. Parameters had to be mapped as well (figure 4.24).

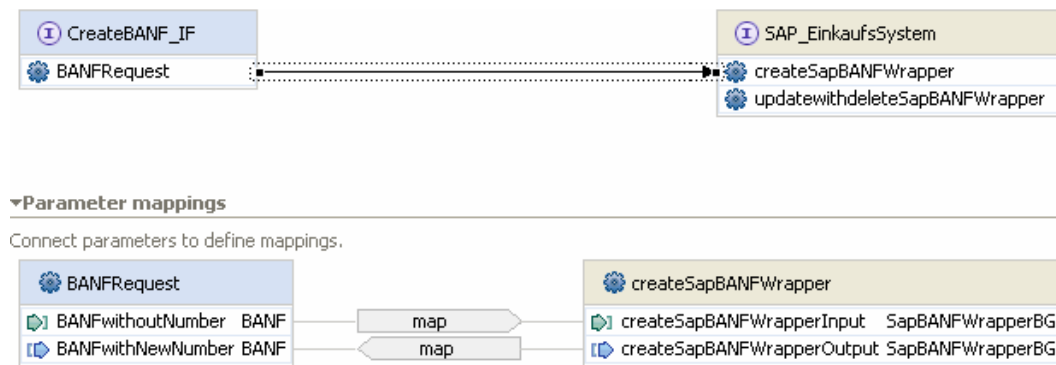


Figure 4.24: Operation and parameter mapping

Because of the complex interface structure of the SAP system, several submaps were necessary to map the concrete parameters to each other in a concise way as shown in figure 4.25.

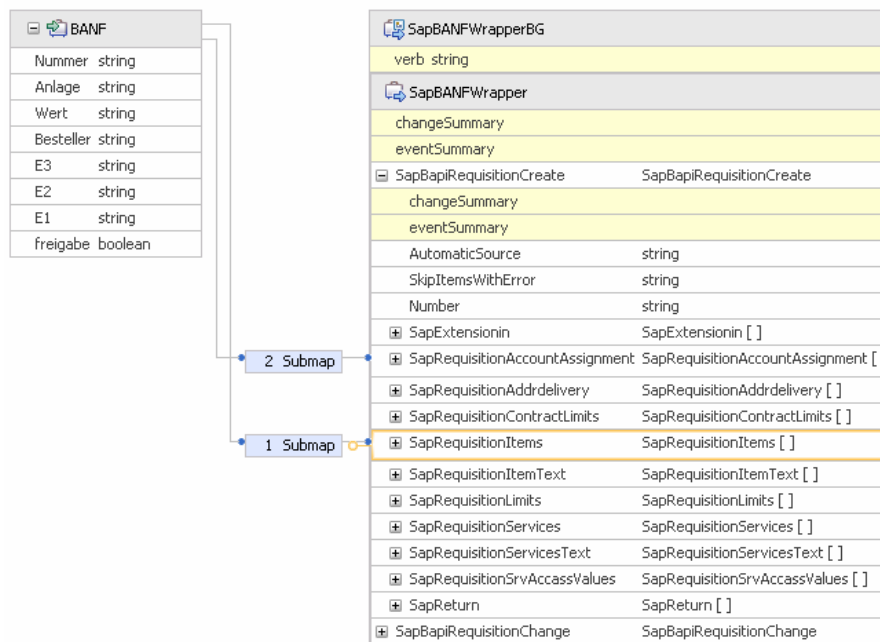


Figure 4.25: Parameter mapping with submaps

Figure 4.26 shows a part of the business object submap that maps the object “BANF” (purchase requisition) to the object “SAPRequisitionItems”. Here, the material number of the purchase requisition is mapped to the material field and the name of the orderer is moved to the “CreatedBy”-field of the interface on the SAP system.

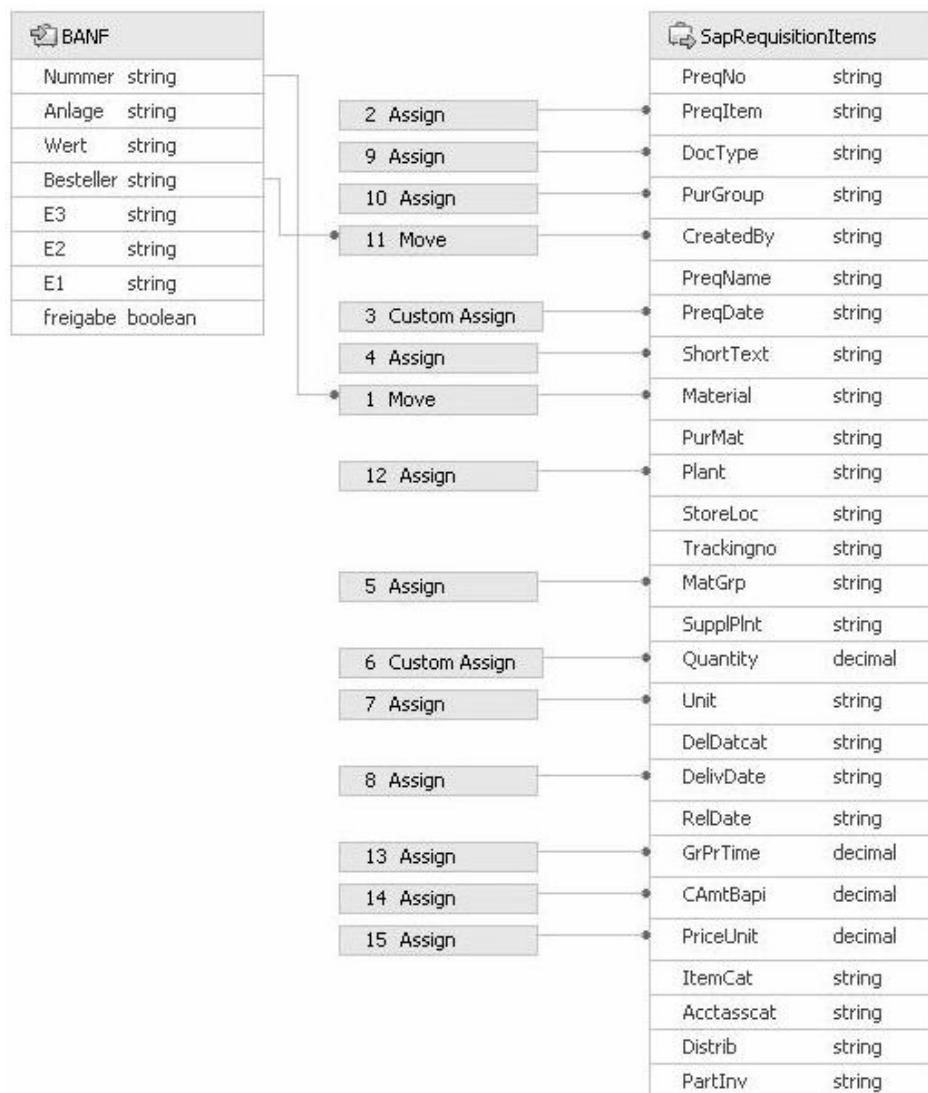


Figure 4.26: Business object mapping

The Update Service

The remote located update service on the SAP system was implemented by generating a Web service interface for an existing application. The resulting Web service without human task (green circle in figure 4.17 on page 71) could be embedded easier in the infrastructure than the application for creating the purchase requisition as no adapter was necessary. However, the service interface was still complex and several mappings were necessary. Figure 4.27 shows the assembly diagram where “Z_BAPI_PR_CHANGE_1Import” represents the external service. “UpdateBANFService” represents a small process where the service is called. An interface map was necessary to map the handed over object to the input object of the small process. The export represents a created Web service binding for the interaction in the assembly diagram. This way, a new Web service was created.

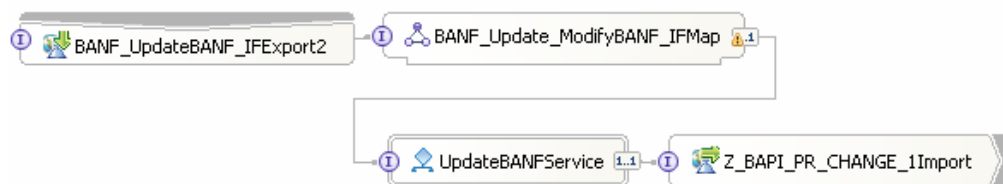


Figure 4.27: Assembly Diagram for the update service

Figure 4.28 shows the business object mapping where id and header text of the purchase requisition are set on the SAP system.

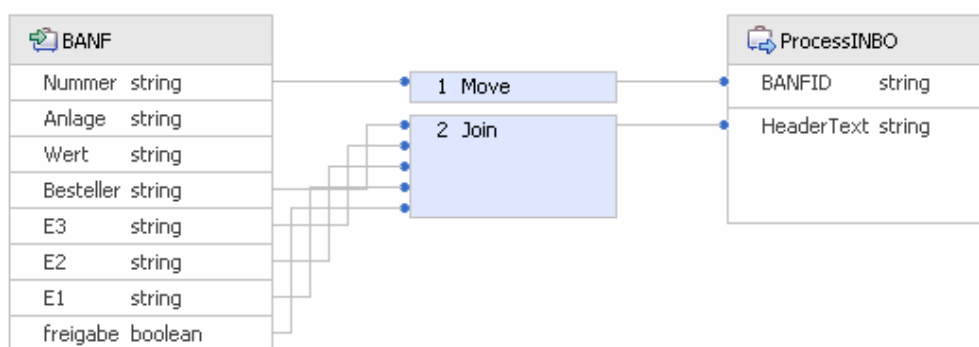


Figure 4.28: Business object mapping

Figure 4.29 demonstrates the small BPEL-process “UpdateBANFService” starting with a *Receive* and ending with a *Reply* element. All developed processes represent again services with input and output parameters. This way, composite services can be created fast. After the assignment of several parameters, the update service is invoked. The *Java Snippet* is responsible for printing out logging information. Another invocation follows where a “commit” for the update service is executed. Also the output parameters for the reply have to be assigned correctly after the invocation.

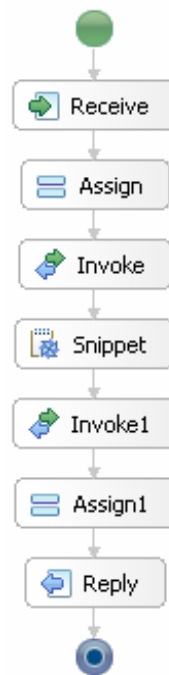


Figure 4.29: Process that invokes the update Web service

4.5 Graphical User Interfaces

Not necessarily a special *Graphical User Interface* (GUI) has to be developed for a process. There is only a need for a graphical surface if process steps include human tasks. Tools for SOA-based integration solutions offer already special GUIs that can be used (i.e. IBM's BPCEXplorer) but usually individual changes concerning functionalities and design are necessary.

A GUI might be recommended for staff members that are not familiar with all the underlying IT-systems. The interface should be clearly arranged and easy to understand. WebSphere Integration Developer offers a possibility of generating a user interface for a developed process. This user interface depends on input and output parameters and JavaServer Pages with JavaServer Faces are generated for the human tasks that take part in the process. The GUI is based on the name of the process and the participating human tasks. This is why it only has to be replaced if the process name or human task settings change. For other modifications of the process, the GUI can be kept and does not have to be regenerated or reimplemented. The GUI can be changed and extended manually by editing the generated JavaServer Pages.

Up to now, Lotus presentation logic is offered for the purchase requisition process at DaimlerChrysler AG. However, a GUI should be developed for the prototype. The BPCEXplorer turned out to be very good for testing but still there was a need for an own graphical user interface with the corporate design of DaimlerChrysler. Special functions were necessary and some of the fields for the input and output variables should be hidden or filled automatically. The opportunity of generating a graphical user interface for the BPEL process was taken. However, the generated JavaServer Pages with JavaServer Faces components had to be changed afterwards manually to implement all needed functionalities. The changes mainly concerned design, navigation and values of parameters.



Figure 4.30: Login screen

In the GUI for the prototype, a user first logs on his personal website using the login screen shown in figure 4.30. Usernames and passwords are sent encrypted by using *HTTPS* (HyperText Transfer Protocol Secure based on *SSL* (Secure Socket layer)). User names and passwords for authorization could have been retrieved per *LDAP* (Lightweight Directory Access Protocol) but for the prototype, users of the operating system (*Microsoft Windows XP*) were mapped to the users that take part in the deployed process.

After logging in, the user reaches an individual page with the user's picture and his personal to-dos (figure 4.31). The user can also start new processes. Only those processes are shown that the user is authorized to start. By entering the identifier "ISP-SENIOR" (which represents purchase requisitions for consulting) and a requested amount (figure 4.31), the purchase requisition can be created by calling the Web service on the SAP system.

Figure 4.31: Individual page

Once a senior manager (2nd line Manager, DaimlerChrysler AG: E3) logs on his individual website, he will see a created task in his to-do list. He gets information about the started process like the date or the name of the creator (figure 4.32).

Taskname ◇	Beschreibung ◇	Erste Aktivierung ◇	Ersteller ◇
▶ BANF-fachlich-freigeben\$BANF-pruefen-und-freigeben-E3Task		11.06.07 11:37:06	meier
Gefundene Elemente: 1 << Seite 1 von 1 >> Elemente pro Seite: 20 ▼			
▶ Aktualisieren			

Figure 4.32: The created task

After opening the shown task, the senior manager gets more information. The input data, the identifying number, the amount and the orderer of the created purchase requisition are shown (figure 4.33).



Figure 4.33: Opened task

A PDF can be opened for a detailed description of the requisition. However, a short example PDF should be sufficient for the prototype (figure 4.34).

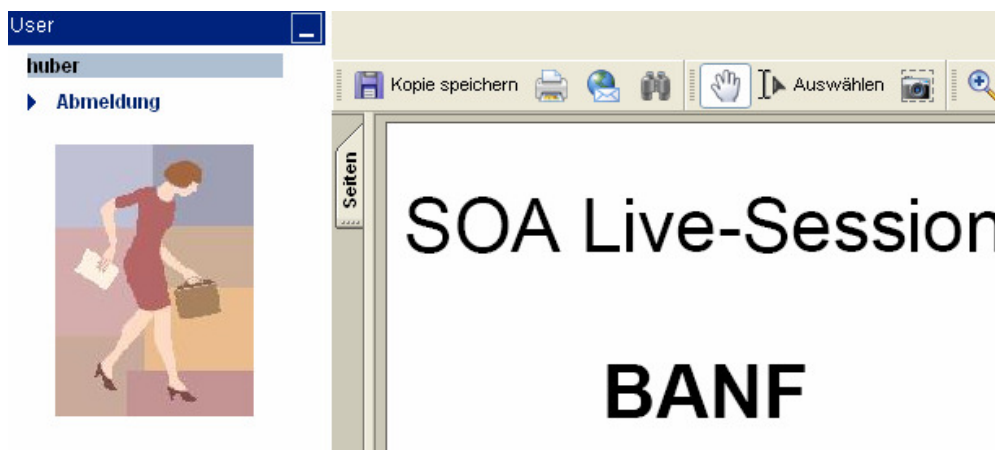


Figure 4.34: Opening the linked PDF

The task can be claimed and a transaction is started. This way, no other user has access to this process until the user is done. There is also support for unclaiming tasks in generated GUIs.

The senior manager can decide if the requisition is approved or declined. After a decline, the process finishes and the purchase requisition on the SAP system is updated with decline information. In case of approval, different cases are possible. The process comes to an end and the purchase requisition is updated if the amount does not exceed 150.000 Euro. In the other case, the next instance gets informed by a new entry in the to-do list. A director (3rd line Manager, DaimlerChrysler AG: E2) has the right to approve if requisition amounts are smaller than or equal to 500.000 Euro by following the same steps. The last instance is represented by a vice president (4th line Manager, DaimlerChrysler AG: E1) who can approve requisitions with higher amounts. Amounts with higher values than 5.000.000 Euro are declined automatically. Figure 4.35 shows the information that the last instance gets about the process.



Figure 4.35: Task of a vice president

As output values have to be filled manually by default, there was a need for setting several variables. Listing 5.2 demonstrates the way input data can be accessed and how the output data is set automatically. Also the current username is saved to the output data in order to keep track of all approvers.

Listing 4.3: Setting output data

```

1 FacesContext context = FacesContext.getCurrentInstance();
2 String user = context.getExternalContext().getRemoteUser();
3 HashMap data = toDoMessageHandler.getToDoInstance().getInputValues();
4
5 if (data.get("/E3") == null)
6 data.put("/E3", user);
7 else if (data.get("/E2") == null)
8 data.put("/E2", user);
9 else if (data.get("/E1") == null)
10 data.put("/E1", user);
11
12 toDoMessageHandler.getToDoInstance().setOutputValues(data);
13 System.out.println("OutputValues:");
14 System.out.println(toDoMessageHandler.getToDoInstance().getOutputValues());

```


The Boolean value for the approval decision was represented by a generated checkbox. This checkbox was replaced by buttons (“Erteilen”, “Ablehnen”) and for each button a different function should be called. For the function for approving purchase requisitions, the Boolean “true” had to be set for the output data as shown in listing 4.4.

Listing 4.4: Setting the Boolean in the approval function

```
1 HashMap data = toDoInstance .getOutputValues();  
2 data.put("/freigabe", "true");  
3 toDoInstance.setOutputValues(data);
```

The modification of the generated JavaServer Pages was necessary to create a concise design with a comprehensible navigation and functionality. The modified GUI represented the purchase requisition process in an understandable way and it helped to demonstrate the prototype.

4.6 Monitoring

The Websphere Process Server offers already a tool for gathering information and monitoring process states, activities and results in the Business Process Container. This tool is called the *BPCObserver*. It was activated in the development phase of the project. Also big monitoring software is developed like the *WebSphere Business Monitor*. However, small monitoring tests were sufficient for this project. The figures 4.36 and 4.37 demonstrate the results of process tests. The color blue represents active processes, red stands for completed processes and green demonstrates the number of failed processes.

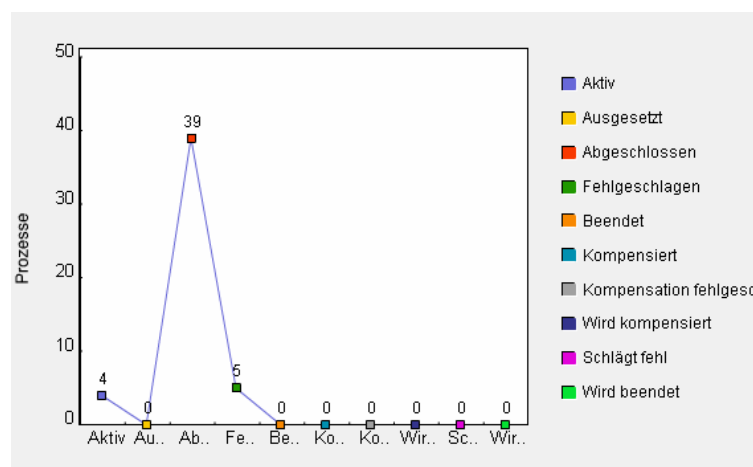


Figure 4.36: Diagram of the BPCObserver for test results of the purchase requisition project

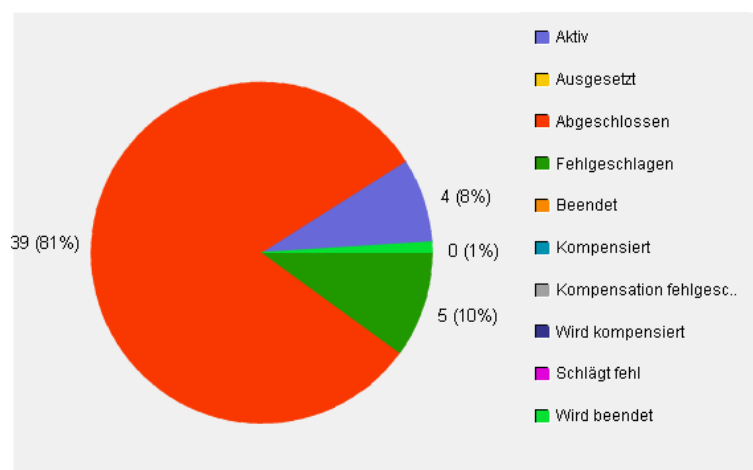


Figure 4.37: Circle diagram of the BPCObserver for test results of the purchase requisition project

5 Change Request

5.1 Strategy

Because of changes of requirements, recommended process improvements or new circumstances, a change request for a process might be necessary. Process steps can change very often and Service-Oriented Architectures shall help for a faster implementation of modifications. Much money and effort could be saved for companies if flexibility and agility concerning process changes were achieved.

Many modifications of the business model are possible like additional process steps, changes of the order of process steps or new called services. The changes should be verified in detail before a change request is submitted. In general, all the steps that were necessary for the implementation of the existing model have to be done again for the modified one. However, based on the existing verified business model, the implementation can be achieved much faster.

For the realization of a process change, the business process model has to be modified involving a change of the BPEL-code that is regenerated. This way, the model is also kept up to date. The change should be documented and the reason for the change should be described, too. The documentation can help the responsible IT specialists for a better understanding of the process change that has to be implemented.

Every modeled process can also be interpreted as a new service for other processes. Services should be registered and all meta information about the services should be found in a repository. For process changes, a possible versioning would make sense where different versions of processes are kept and are callable. When editing the existing business model, equal sub processes should be searched and also already existing projects with similar goals should be checked for relevance. Eventually, services already exist that can be reused.

Modifications of a business model can involve small but also big changes of the whole process implementation. The modeled process modification has to be discussed based on the eEPC in order to ensure that the responsible IT specialists understand the process change. Because of the high level of abstraction that can be reached with eEPCs, the model should be understandable for both, the business unit and the IT developer. The business model should be adequate for a common basis for discussion.

Ideally, a merge of the old implementation with the modification should be possible in order to get faster implementation results. As WebSphere Integration Developer does not yet support merging old versions with modified BPEL, the implementation steps have to be done again.

5.2 Practical Example

Incoming change requests for processes usually cost a lot of effort and are money- and time- consuming for a company. The example process should be modified in order to demonstrate the agility and flexibility of a Service-Oriented Architecture and possible cost savings. Incoming change requests address changes concerning process steps of a business model. They can include demands for new Web services or the use of already existing ones. In the beginning of a SOA infrastructure development, mostly new services will be needed until a repository of reusable services is built. New required services have to be implemented or old applications with the needed functionality are wrapped with a usable interface.

For the example process, a reasonable change should be carried out. In the existing model, the orderer and the managers are not informed about approved or declined requisitions. The planned process modification should address this problem. In case of denial, ideally every predecessor should get an e-mail of the next higher instance with information about the declined requisition. This way, all participants get informed.

For demonstration purposes, the process change should not be that time-consuming and this is why only one change should be carried out. An e-mail should be sent from the senior manager to the orderer once the senior manager declines a purchase requisition. As all the other changes would be modeled the same way, only this single change of the process model should be shown. The new e-mail service was implemented locally and was running on the integrated Process Server (figure 5.1).

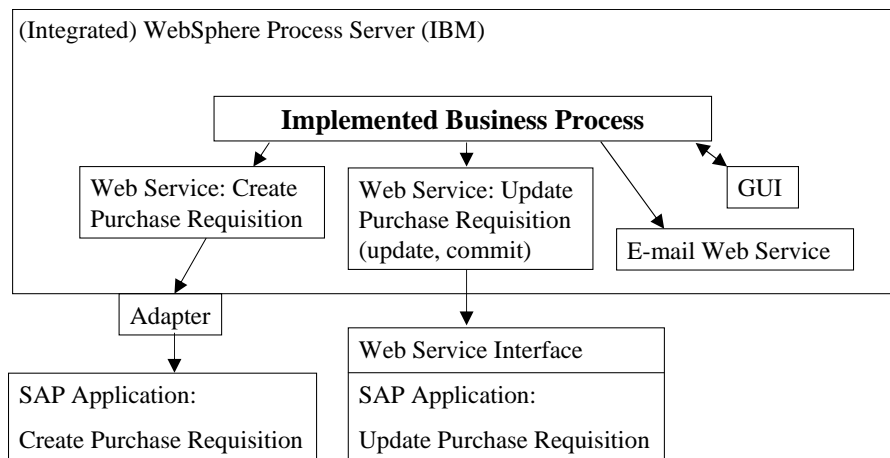


Figure 5.1: The new e-mail service

Figure 5.2 shows the resulting eEPC after the modification of the example process model. A new Web service is called after the decline of a purchase requisition by a senior manager. The business object "BANF" with all its values represents the input for a process with the name "Vorgaenger informieren" that calls the Web service. The Web service is responsible for sending an e-mail to the orderer that initiated the process. The function is followed by an event for getting a correct and convertible eEPC.

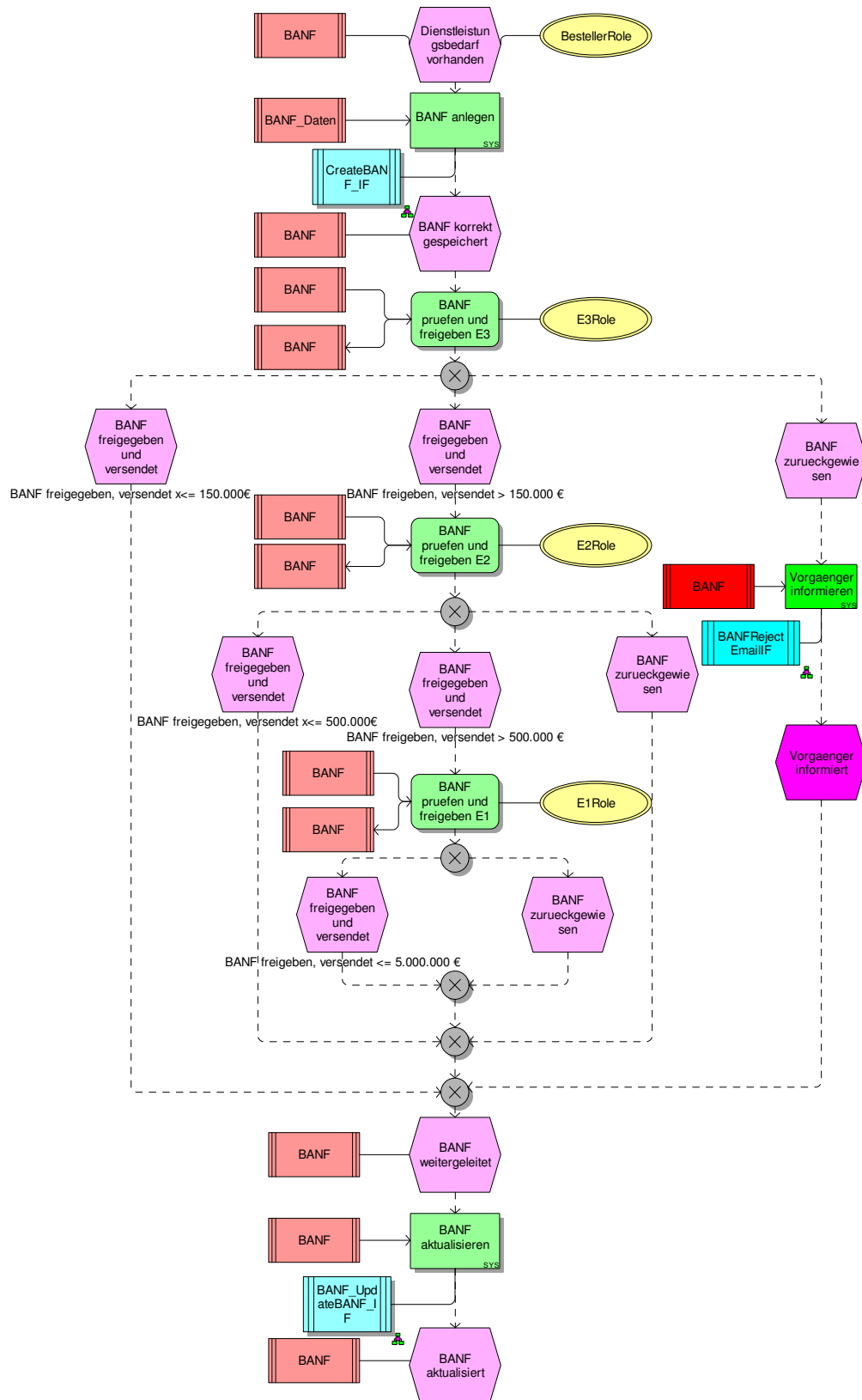


Figure 5.2: EPC of the business model with process modification

Again, BPEL-code was generated with EPC2BPEL and the two proprietary scripts prepared the BPEL-code for the import in WebSphere Integration Developer. For the example project, a new orchestrated Web service without human tasks was requested. The Web service should take care of sending e-mails and was implemented directly in WebSphere Integration Developer. The change of the BPEL-code turned out to be small. A partner link was added and the special changed case was specified. An invocation for the Web service that references the new partner link was also added. Both changes of the BPEL-code are shown in listing 5.1.

Listing 5.1: Changes in the BPEL-code

```
1 <partnerLink xmlns:imp1="http://EmailService/BANFRejectEmailIF/Binding2"
2 name="BANFRejectEmailIFPL"
3 partnerLinkType="imp1:BANFRejectEmailWS_BANFRejectEmailIFHttpServicePLT"
4 partnerRole="requester" />
5
6 <case
7 condition="bpws:getVariableData('BANF','freigabe')
8 =false()">
9     <invoke xmlns:imp1="http://EmailService/BANFRejectEmailIF"
10     inputVariable="BANF"
11     name="Vorgaenger-informieren"
12     operation="info"
13     partnerLink="BANFRejectEmailIFPL"
14     portType="imp1:BANFRejectEmailIF"
15     />
16 </case>
```

After the import of the modified BPEL-code, a new invocation symbol appeared in the *Business Process Editor*. The symbol is located directly after the case check for the decline of the senior manager and is shown in figure 5.3.

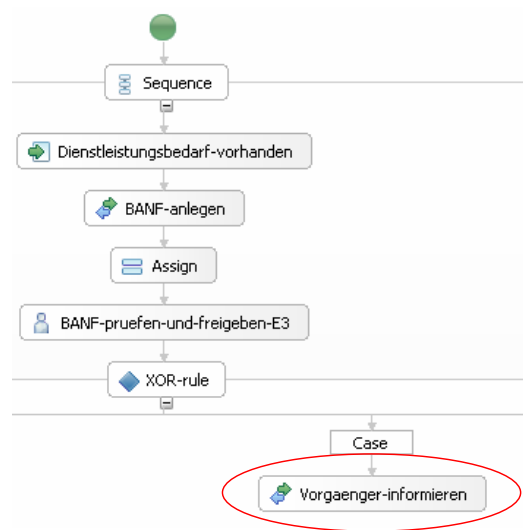


Figure 5.3: Modified BPEL in the Business Process Editor

Adjustments had to be made again in order to make the process deployable on the integrated Process Server. The WSDL file for the new Web service had to be saved in the library because of problems with the remote import. All adoptions had to be redone for the whole process including the new service call as no merge was possible. Dependencies had to be reset, data type variables had to be used and correct reference partners had to be selected. Also missing human task information had to be added again. For the new Web service, only small changes were necessary like also choosing the correct reference partner with the right operation and data type variables.

The E-mail Service

The service itself was implemented by creating an own new project in WebSphere Integration Developer. Figure 5.4 shows the Assembly Diagram of the e-mail Web service where the generated Web service binding “BANFRejectEmailWS” represents the interface.



Figure 5.4: E-mail service in the Assembly Diagram

An interface mapping was necessary to map the incoming business object “BANF” to the object “EmailMessage” (figure 5.5).



Figure 5.5: Interface mapping for the e-mail service

The business object mapping is shown in figure 5.6 where information about number, amount, orderer, approvers and result of the purchase requisition is combined to build the body of the e-mail. Subject, sender and receiver were hard-coded by assignments for the prototype.

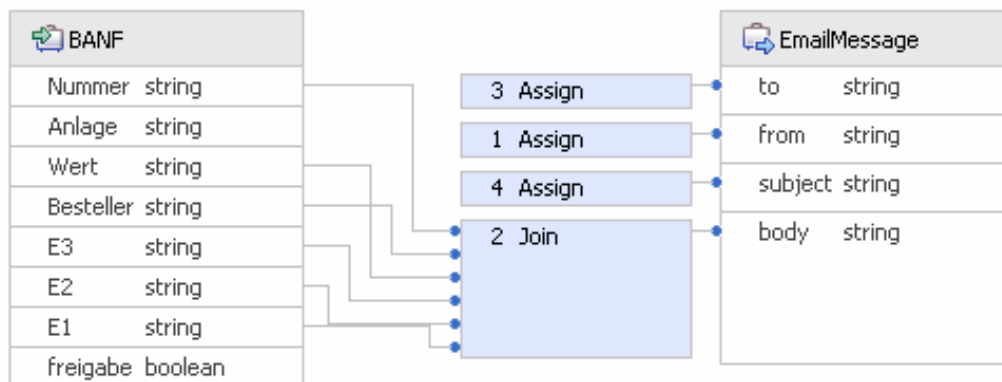


Figure 5.6: Business object mapping for the e-mail service

A part of the implementation of the e-mail service is shown in listing 5.2. There, the necessary variables are set with the information that is provided by the business object mapping.

Listing 5.2: Implementation of the e-mail service

```
1 public void send(DataObject input1) {  
2     String to = input1.getString("to");  
3     String from = input1.getString("from");  
4     String subject = input1.getString("subject");  
5     String body = input1.getString("body");  
6     System.out.println("E-mail_body:_");  
7     System.out.println(body);  
8     Send send = new Send();  
9     send.send(to, from, subject, body);  
10 }
```

Like all the other used Web services, the new e-mail Web service had to be deployed on the (integrated) Process Server for being available for the main process. This way, an e-mail notification about declined purchase requisitions could be sent to the creator of the purchase requisition.

6 Summary and Conclusion

6.1 Results of the SOA Live Session Project

With the SOA Live Session project, much experience has been gained concerning the ideas of SOA, its benefits and its drawbacks. The prototype helped to provide an insight into SOA and the current state of maturity of existing tools. The resulting working project demonstrated that processes can indeed be implemented with this SOA approach. The explained top-down strategy was realized and experiences could be gained.

In addition, the process was modified to demonstrate the flexibility and agility that can be achieved with a SOA. An ad hoc implementation of a process change was demonstrated. The process modification was carried out based on the top-down approach starting with a change of the business process model.

Software of different vendors (IDS Scheer AG, IBM and SAP) was used for the process implementation. This proved the possibility of combining heterogeneous tools and services in a SOA. WSDL (Web Services Description Language) interfaces were developed for external SAP applications in order to enable their successful integration.

During the project, an insufficient state of maturity of SOA development software was recognized. Compatibility problems among software of different vendors were found and caused implementation problems. A solution of these compatibility problems was developed in close cooperation with the vendors. Because of the immaturity of the existing BPEL standard, vendors developed proprietary BPEL extensions and compatibility problems were based on these vendor specific differences.

The prototype was presented to the executive management at DaimlerChrysler AG and valuable feedback comments could be gained. Questions were raised about time and effort that was needed to implement the process following the top-down approach. Although the process was small and IBM Deutschland GmbH, IDS Scheer AG and SAP AG supported the project, three months were necessary to create the prototype. Certainly, this project was the first SOA project and experience had to be collected first. However, a significant amount of effort was required based on restrictions related to the maturity level of the tools used and the compatibility among each other. The top-down approach was demanding for the tools of different companies on different layers. Because of the restricted maturity of tools, business critical processes should not yet be implemented based on a Service-Oriented Architecture. On

the other hand, as a joint effort with companies like IBM Deutschland GmbH, IDS Scheer AG and SAP AG, the planned project could be successfully implemented and the fast modification of the process could be demonstrated. The companies already started to improve their products based on the problems that arised and based on the feedback they received. The reliability of the tools will certainly be improved involving a faster development.

One conclusion is the fact that SOA projects need a lot of time and investment before profiting from efforts. First projects will be both time- and money-consuming until a considerable amount of services is reached (later Return on Investment).

Another discussed topic concerned the way business process models are created. Event-driven Process Chains represent a very good base for common discussions between the business unit and IT developers. They are relatively easy to read and understand. However, still several rules have to be considered for creating a convertible business model. Responsible persons would have to be trained in creating convertible models. Various rules are already explained in this thesis but still more experience is needed for defining rules for any kind of business process for a company.

As a result of the work performed in this thesis, DaimlerChrysler decided to start further SOA projects and to continue researching into SOA and its relevance and influence on the company.

6.2 Conclusion

Although the term SOA is mentioned very often in the media and many companies develop and do research for SOA solutions, much additional tool development has to occur before Service-Oriented Architectures become a main stream approach for heterogeneous environments. However, many people and companies expect a lot from this new architecture and hope that this way, a better control over heterogeneous IT landscapes can be achieved. With the project of this thesis, a first step for creating compatibility between SOA software of different companies could be made, the current state of developing SOAs could be analyzed and several still existing problems could be exposed. The development of further standards will hopefully make cooperation easier between companies.

The prototype did not require an Enterprise Service Bus and also repositories were not accessed. Future work should also include these components in the architecture. With an ESB, also mediation implementations should be analyzed. Certainly, several repositories of different vendors will be used in the future and the cooperation of different repositories will be necessary. Many compatibility issues will have to be addressed and also security aspects still represent a big challenge. SOA governance will also play a decisive role for the success of SOA projects in companies.

List of Figures

2.1	Expected benefits of SOA [Pez06]	9
2.2	Expected drawbacks of SOA [Pez06]	10
2.3	Orchestration And Choreography [Erl05]	14
2.4	The Find, Bind, and Execute Paradigm [MTSM03]	19
2.5	Big Picture SOA [Dai07]	21
2.6	Enterprise Service Bus and connected component types[Bal05]	24
2.7	Models of Interaction between Tasks and Processes [III ⁺ 05]	35
3.1	Generic proceeding (top-down)	38
3.2	EPC: Event symbol [DB07]	40
3.3	EPC: Function symbol [DB07]	40
3.4	EPC: Organisation unit symbol [DB07]	42
3.5	EPC: Data model symbol [DB07]	42
3.6	EPC: Web service symbol [DB07]	42
3.7	EPC: XOR, AND, OR [DB07]	43
3.8	Logical Operators [DB07]	43
3.9	An example eEPC [KNS92]	44
3.10	Web service call example	45
3.11	Service overview [Dai07]	50
4.1	Application architecture for a Consumer Electronics Company [Spr07]	51
4.2	Proceeding (top-down)	53
4.3	Acquisition process	55
4.4	Purchase requisition process description	57
4.5	Draft eEPC	59
4.6	Legend for figure 4.5	60
4.7	The final eEPC of the business process	62
4.8	Legend for figure 4.7	63
4.9	Business process	64
4.10	UML representation of the data object "BANF"	64
4.11	Development procedure	66
4.12	Human task extension in the BPEL allocation diagram	67
4.13	A completed BPEL allocation diagram	68
4.14	Generated port type with operations	68
4.15	Graphical demonstration of the generated BPEL-code in ARIS SOA Architect	69
4.16	Group symbols	70
4.17	Service implementation overview [Dai07]	71
4.18	Used business object	72

4.19	BPEL process in WID (1)	73
4.20	BPEL process in WID (2)	73
4.21	Service calls	74
4.22	Service calls with extra services	74
4.23	Interface map in the assembly diagram	75
4.24	Operation and parameter mapping	76
4.25	Parameter mapping with submaps	76
4.26	Business object mapping	77
4.27	Assembly Diagram for the update service	78
4.28	Business object mapping	78
4.29	Process that invokes the update Web service	79
4.30	Login screen	81
4.31	Individual page	82
4.32	The created task	82
4.33	Opened task	83
4.34	Opening the linked PDF	83
4.35	Task of a vice president	84
4.36	Diagram of the BPCObserver for test results of the purchase requisition project	86
4.37	Circle diagram of the BPCObserver for test results of the purchase requisition project	86
5.1	The new e-mail service	89
5.2	EEPC of the business model with process modification	90
5.3	Modified BPEL in the Business Process Editor	92
5.4	E-mail service in the Assembly Diagram	92
5.5	Interface mapping for the e-mail service	93
5.6	Business object mapping for the e-mail service	93
A.1	E-mail with information about the purchase requisition process	109
A.2	Purchase Requisition Information	110
A.3	BPEL process in WID (1)	111
A.4	BPEL process in WID (2)	111
A.5	BPEL process in WID (3)	112
A.6	Aris EPC rules (structure rules) [AG06]	113
A.7	Aris EPC rules (Rules for service-oriented EPC 1) [AG06]	114
A.8	Aris EPC rules (Rules for service-oriented EPC 2) [AG06]	115
A.9	CD Content	116

List of Tables

3.1	Abstraction layer [KNS92]	41
3.2	Specification layer [KNS92]	41
3.3	General EPC rules [DB07]	46
3.4	Structure rules for EPCs [AG06]	47
3.5	Rules for service-oriented EPCs [AG06]	48
A.1	Acronym descriptions for the acquisition process on page 55	108

Listings

2.1	BPEL process [Dje04]	28
2.2	Definition of partnerLinkTypes [Dje04]	29
2.3	The <partners> [Dje04] element	29
2.4	Invoking the operation “buy” with the input variable “itemid” [Dje04]	29
2.5	The <receive> element [Dje04]	30
2.6	With the <reply> element a price is returned to the caller [Dje04]	30
2.7	Sequence for a synchronous Web service call [Dje04]	30
2.8	Flow example for executing A before B [Dje04]	31
2.9	Variable definition to an xsd type int [Dje04]	31
2.10	Assigning a value to a variable [Dje04]	31
2.11	Defining a property for enabling public access to an attribute [Dje04]	32
2.12	Defining a correlationSet for the user identification [Dje04]	32
4.1	Condition expression for branches	65
4.2	Human task extension in BPEL-code	70
4.3	Setting output data	84
4.4	Setting the Boolean in the approval function	85
5.1	Changes in the BPEL-code	91
5.2	Implementation of the e-mail service	94

Acronyms

Notation	Description
ARIS	Architecture of Integrated Information Systems
BAPI	Business Application Programming Interface
BPCExplorer	Business Process Choreographer Explorer
BPCObserver	Business Process Choreographer Observer
BPEL4WS	Business Process Execution Language for Web Services
CORBA	Common Object Request Broker Architecture
DCE	Distributed Computing Environment
DCOM	Distributed Component Object Model (Microsoft)
EAI	Enterprise Application Integration
EAR	Enterprise Application Archive
EJB	Enterprise Java Beans
ESB	Enterprise Service Bus
GLOBUS	Global Buying System
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
J2EE	Java 2 Enterprise Edition
JCo	Java Connector
JMS	Java Message Service
JSF	JavaServer Faces
JSP	JavaServer Pages
LDAP	Lightweight Directory Access Protocol

Notation	Description
NACOS	New Accounting and Controlling System
OASIS	Organization for the Advancement of Structured Information Standards
PDF	Portable Document Format
QoS	Quality of Service
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SOA	Service-Oriented Architecture
SOAP	First name: Simple Object Access Protocol, later also known as the Service-Oriented Architecture Protocol. In order to dissociate from these descriptions, it is just called SOAP today.
SQL	Structured Query Language
SSL	Secure Socket layer
UDDI	Universal Description, Discovery and Integration. UDDI defines a set of services that support the description and discovery of organizations, businesses and other Web service providers, their published Web services and the technical interfaces for accessing the Web services
UML	Unified Modeling Language
VCD	Value Chain Diagrams
W3C	World Wide Web Consortium
WID	WebSphere Integration Developer
WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Service Description Language
XML	Extensible Markup Language
XPath	XML Path Language
XQuery	XML Query Language
XSD	XML Schema Definition

Bibliography

- [ACG⁺03] T. Andrews, F. Curbera, H. Dholakia and Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. <ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf>, 2003.
- [AG06] IDS Scheer AG. *ARIS SOA Architekt*, 1997-2006.
- [Bal05] Naveen Balani. Model and build ESB SOA frameworks. <http://www-128.ibm.com/developerworks/web/library/wa-soaesb/>, 2005.
- [BBF⁺06] Norbert Bieberstein, Sanjay Bose, Marc Fiammante, Keith Jones, and Rawn Shah. *Service-Oriented Architecture (SOA) Compass*. Pearson Education, Inc., Rights and Contracts Department. One Lake Street. Upper Saddle River, NJ 07458, 2006.
- [BM03] Colin Boyd and Wenbo Mao, editors. *Information Security: 6th International Conference, Isc 2003, Bristol, Uk, October 2003 Proceedings*. Springer, 2003.
- [CG00] Robert Cailliau and James Gillies. *How the Web Was Born: The Story of the World Wide Web*. Oxford University Press, 2000.
- [Col04] Mark Colan. Service-Oriented Architecture expands the vision of Web services, Part 1, Characteristics of Service-Oriented Architecture. <http://www-128.ibm.com/developerworks/library/ws-soaintro.html>, 2004.
- [Dai07] DaimlerChrysler. ITP/AM Technology and Methods MCG, 2007.
- [DB07] Rob Davis and Eric Brabänder. *ARIS Design Platform. Getting Started with BPM*. Springer Verlag, London, 2007.
- [Dje04] Riad Djemili. BPEL4WS. Business Process Execution Language for Web Services, 2004.
- [DJMZ05] Wolfgang Dostal, Mario Jeckle, Ingo Melzer, and Barbara Zengler. *Service-orientierte Architekturen mit Web Services. Konzepte - Standards - Praxis*. Elsevier GmbH, München, 2005.
- [DRS⁺07] Chris Dudley, Laurent Rieu, Martin Smithson, Tapan Verma, and Byron Braswell. *WebSphere Service Registry and Repository Handbook. IBM Document No. SG24-7386-00*, 2007.

- [Erl05] Thomas Erl. *Service-Oriented Architecture, Concepts, Technology, and Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Pearson Education, Inc., Rights and Contracts Department. One Lake Street. Upper Saddle River, NJ 07458, 2005.
- [FM02] Alex Ferrara and Matthew MacDonald, editors. *Programming .NET Web Services*. O'Reilly, 2002.
- [GG06] Javier Garcia and German Goldszmidt. Building SOA composite business services, Part 1: Develop SOA composite applications to enable business services. <http://www.ibm.com/developerworks/webservices/library/ws-soa-composite/>, 2006.
- [Ghe97] Iosif G. Ghetie. *Networks and Systems Management: Platforms Analysis and Evaluation*. Springer, 1997.
- [GSM02] Dominik Gruntz, Clemens Szyperski, and Stehpan Murer. *Component Software Beyond Object-Oriented Programming*. ACM Press, New York, 2002.
- [GYVN03] Roy W. Schulte Gartner: Yefim V. Natis. *Introduction to Service-Oriented Architecture*, 2003.
- [Har04] Elliotte Rusty Harold. *Java Network Programming*. O'Reilly, 2004.
- [HB04] Hugo Haas and Allen Brown. W3C Working Group Note, Web Services Glossary. <http://www.w3.org/TR/ws-gloss/>, 2004.
- [III⁺05] Matthias Kloppmann (IBM), Dieter Koenig (IBM), Frank Leymann (IBM), Gerhard Pfau (IBM), Alan Rickayzen (SAP), Claus von Riegen (SAP), Patrick Schmidt (SAP), and Ivana Trickovic (SAP). *WS-BPEL Extension for People – BPEL4People*. A Joint White Paper by IBM and SAP, 2005.
- [JE07] Diane Jordan and John Evdemon. OASIS Web Services Business Process Execution Language Version 2.0, Committee Specification. <http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.pdf>, 2007.
- [KAH⁺05] Martin Keen, Oscar Adinolfi, Sarah Hemmings, Andrew Humphreys, Hanumanth Kanthi, and Alasdair Nottingham. *Patterns: SOA with an Enterprise Service Bus in WebSphere Application Server V6*. IBM Document No. SG24-6494-00, 2005.
- [KBS05] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA. Service-Oriented Architecture Best Practices*. Prentice Hall Professional Technical Reference, New Jersey, 2005.
- [Kla06] Prof. Dr. Herbert Klaeren. *Skriptum Softwaretechnik*, 2006.
- [KNS92] Gerhard Keller, Markus Nüttgens, and August-Wilhelm Scheer. *Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter*

- Prozeßketten (EPK). In *August-Wilhelm Scheer (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89*. <http://www.iwi.uni-sb.de/Download/iwihefte/heft89.pdf>, Saarbrücken, 1992.
- [Kra06] Volker Kramberg. Pattern-based Evaluation of IBM WebSphere BPEL, 2006.
- [LA98] Peter Loos and Thomas Allweyer. Process Orientation and Object-Orientation - An Approach for Integrating UML and Event-Driven Process Chains (EPC), Paper 144. Institut für Wirtschaftsinformatik, University of Saarland, Saarbrücken, 1998.
- [LR00] Frank Leymann and Dieter Roller. *Production Workflow (Concepts and Techniques)*. Prentice Hall PTR, Upper Saddle River, New Jersey 07458, 2000.
- [MTSM03] James McGovern, Sameer Tyagi, Michael E. Stevens, and Sunil Mathew. *Java Web Services Architecture*. Morgan Kaufmann, San Francisco, 2003.
- [Nüb07] Marc Nübling. Entwurf einer Security-Infrastruktur für SOA auf Basis von Web Services und IBM WebSphere Developer, 2007.
- [OAS04] OASIS. UDDI Version 3.0.1. <http://uddi.org/pubs/uddi-v3.0.1-20031014.pdf>, 2004.
- [OMG07] Inc. Object Management Group. CORBA BASICS. <http://www.omg.org/gettingstarted/corbafaq.htm>, 2007.
- [OWRB06] Kai J. Oey, Holger Wagner, Simon Rehbach, and Andrea Bachmann. Mehr als alter Wein in neuen Schläuchen. Eine einführende Darstellung des Konzepts der serviceorientierten Architekturen. In *Unternehmensarchitekturen und Systemintegration*, page 197 ff. Gito, Berlin, 2006.
- [Pez06] Massimo Pezzini. An SOA Maturity Model: Where Do You Stand and Where Are You Going?, 2006.
- [SCD00] Darleen Sadoski and Santiago Comella-Dorda. Three Tier Software Architectures. Software Technology Roadmap. http://www.sei.cmu.edu/str/descriptions/threetier_body.html, 2000.
- [Sch00] August-Wilhelm Scheer. *ARIS - Business Process Modeling*. Springer Verlag, Berlin, third edition, 2000.
- [Sei02] Heinrich Seidlmeier. *Prozessmodellierung mit ARIS. Eine beispielorientierte Einführung für Studium und Praxis*. Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 2002.
- [Shi03] Robert J. Shimonski. *Building DMZs For Enterprise Networks*. Syngress Publishing, 2003.
- [Sie05] Johannes Siedersleben. *Moderne Softwarearchitektur*. dpunkt.verlag, Heidelberg, 2005.

- [Spr07] Prof. Dr.-Ing. Wilhelm G. Spruth. Personal communication, 2007.
- [SW01] Uwe Schneider and Dieter Werner. *Taschenbuch der Informatik*. Fachbuchverlag Leipzig im Carl Hanser Verlag München Wien, 2001.
- [Szy02] Clemens Szyperski. *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley, 2002.
- [vHOS05] Kees van Hee, Olivia Oanea, and Natalia Sidorova. Colored Petri Nets to Verify Extended Event-Driven Process Chains. Department of Mathematics and Computer Science, Eindhoven University of Technology, 2005.
- [WCL⁺05] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donal F. Ferguson. *Web Services Platform Architecture. SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Pearson Education, Inc., Rights and Contracts Department. One Lake Street. Upper Saddle River, NJ 07458, 2005.
- [WM06] Dan Woods and Thomas Mattern. *Enterprise SOA - Designing IT for Business Innovation*. O'Reilly Media, 2006.
- [Zül04] Heinz Züllighoven. *Object-Oriented Construction Handbook: Developing Application-oriented Software with the Tools and Materials Approach*. Morgan Kaufmann Publishers Inc, US, 2004.

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Oliver Dalferth

Tübingen, September 1, 2007

Appendix A: Process Descriptions

Appendix A provides acronym descriptions for the acquisition process shown on page 55. Information is given that was necessary to understand the process steps of the implemented purchase requisition process. An e-mail and a PDF with process information is shown. Also the whole graphical interpretation of the completed BPEL process in WebSphere Integration Developer is attached.

Acronym	German	English
MAB	Materialanforderungsbeleg	Material requisition note
BNS	Benachrichtigungsschein	Notification certificate
MBD	Material-Bestandsführung und Disposition	Material Inventory Maintenance and Disposition
IDS	Instandhaltungs- und Ersatzteile- Dokumentationssystem	Maintenance and Spares Documentation System
NACOS	Neues Accounting- und Controlling-System	New Accounting and Controlling System
MES	Material-Einkauf-System	Material Buying System
MABB	Material-Anforderungsbeleg-Buchung	Material requisition receipt book entry
ZB	Zentrale Buchung	Central entry
WBS	Warenbegleitschein	Stock dispatch note
WES	Wareneingangs und -abgleich-system	Stock Receipt and Reconciliation system

Table A.1: Acronym descriptions for the acquisition process on page 55

Liebe BANF-Aussteller,

wie in der Mail vom 20.01.2006 angekündigt wird ab morgen der Workflow für BANFen > 25 TE aktiv sein. Hier die Basics in Kürze:

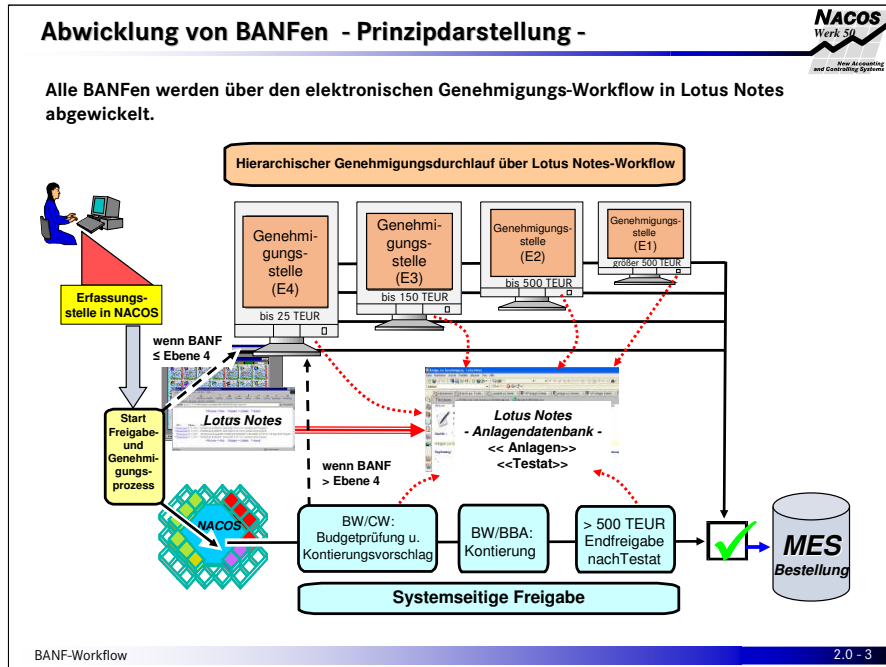
Achtung: Die Änderungen gelten nur für BANFen, die ab dem 01.02.2006 erstellt werden

- Die Papierabwicklung und -ablage entfällt für alle BANFen, die ab dem 01.02.06 erstellt werden
- Alle Führungskräfte (E4-E1) geben BANFen per Lotus Notes Workflow frei

Ablauf:

- BANFen <= 25 TE wie bisher :
=> AS-Freigabe durch Aussteller
=> Budgetprüfung und EP-Freigabe durch das Controlling / parallel E4-Freigabe
=> Freigabe BBA
=> ZSAB: Abruf bzw. ZNB: zu Einkauf
- BANFen > 25 TE:
=> AS-Freigabe durch Aussteller
=> Budgetprüfung und EP-Freigabe durch das Controlling
=> Freigabe E4
=> Freigabe E3 (...)
=> Freigabe BBA
=> ZSAB: Abruf bzw. ZNB: zu Einkauf

Figure A.1: E-mail with information about the purchase requisition process



Nutzen der elektronischen BANF-Genehmigung:

- Qualitative Verbesserungen:
 - Revisionssichere Dokumentation der Genehmigung (SOA-Anforderungen).
 - Schlanker standardisierter Prozess mit hoher Transparenz.
 - Beschleunigung der Bearbeitungszeit (paralleler Ablauf von hierarchischer Genehmigung im Fachbereich und Freigabe in BBA).
- Einsparungen aufgrund kürzerer Bearbeitungszeit und Reduzierung um über 50.000 Papiervorgänge p.a.
- Die aufwändige Archivierung von ausgedruckten und unterzeichneten BANF-Formularen entfällt.

Figure A.2: Purchase Requisition Information

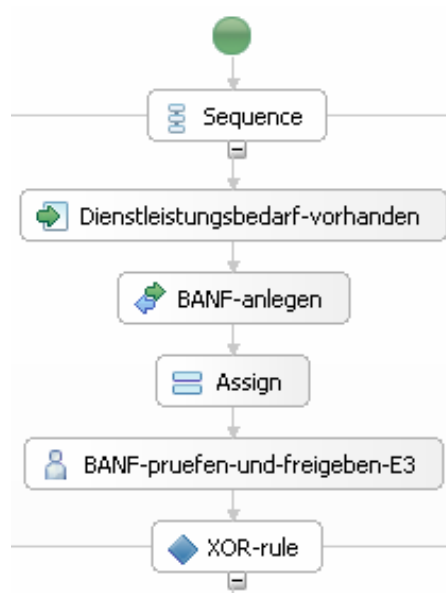


Figure A.3: BPEL process in WID (1)

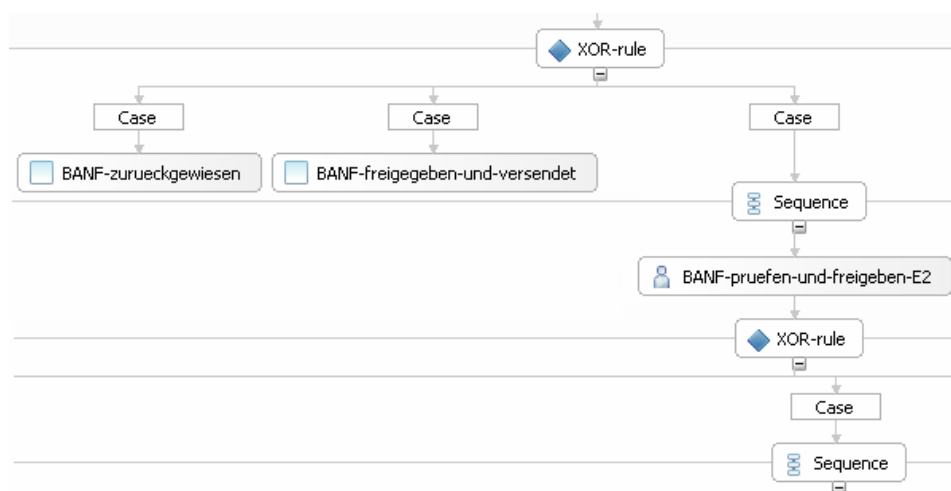


Figure A.4: BPEL process in WID (2)

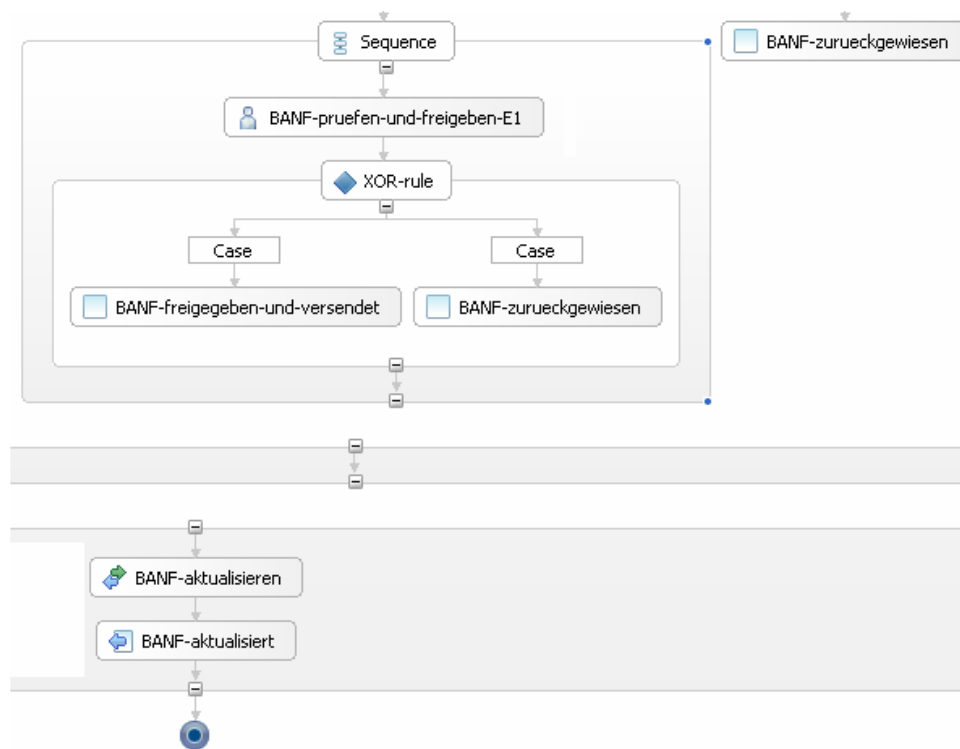


Figure A.5: BPEL process in WID (3)

Appendix B: ARIS Rules for Event-driven Process Chains

ARIS SOA Architect offers a facility to check modeled Event-driven Process Chains for correctness. A page with validation results is generated. Appendix B shows the validation pages of a correct EPC.

ARIS Semantic Check

Validation of a service-oriented EPC

Validates whether a service-oriented EPC is modeled correctly.

Structure rules		
Rule:All functions/events have only one incoming/outgoing connection		
Description:This rule checks whether all functions and events have a maximum of one incoming or outgoing connection.		
The following functions and events have more than one incoming or outgoing connection:		
Checked model	Object name	Object type
Check produced no errors.		
Rule:Each path must begin and end with an event		
Description:This rule checks whether all paths begin and end with an event.		
The following start or target objects are no events:		
Checked model	Object name	Object type
Check produced no errors.		
Rule:No OR/XOR possible after event		
Description:This rule checks whether splitting OR or XOR rules (distributors) do not exist within a process after events.		
The following events have an OR or an XOR rule as successor:		
Checked model	Event	Succeeding rule
Check produced no errors.		
Rule:No objects without connections may exist		
Description:This rule checks whether a model contains object occurrences without connections to other occurrences. Each object in a model must have one or more predecessors and/or successors.		
The following objects have no connections to other objects:		
Checked model	Object name	Object type
Check produced no errors.		
Rule:Number of outgoing or incoming connections at the rule		
Description:This rule checks whether at each simple rule there are either exactly one incoming and a minimum of two outgoing connections, or a minimum of two incoming and exactly one outgoing connection.		
The number of incoming and outgoing connections is not correct for the following rules:		
Checked model	Object name	Object type
Check produced no errors.		

Figure A.6: Aris EPC rules (structure rules) [AG06]

Rules for service-oriented EPC
Model : BANF fachlich freigeben
Rule: A business function should be carried out by one single organizational unit.
Description: Only a business function that is connected with one single object of the organizational unit type via a relationship of the 'carries out' type is interpreted via the transformation to the BPEL process.
Semantic check was successful.
Model : BANF fachlich freigeben
Rule: A system function is supported by one single object of the 'Application system type' type.
Description: A system function may be supported by only one single object of the 'Application system type' type and is thus interpreted via the transformation to the BPEL process.
Semantic check was successful.
Model : BANF fachlich freigeben
Rule: All input and output objects must be mapped to objects of the 'Class' type or be themselves objects of the 'Class' type.
Description: All objects that are connected with functions via relationships of the 'has input' or 'has output' type should be of the 'Class' type or be directly or indirectly mapped to one or more objects of the 'Class' type to be interpreted correctly via the transformation to the BPEL process.
Semantic check was successful.
Model : BANF fachlich freigeben
Rule: An 'Application system type' object supporting a system function may be connected with only one object of the 'Component' type.
Description: According to the modeling conventions for the representation of a service in ARIS, only one single object of the 'Component' type may be connected with an 'Application system type' object via a relationship of the 'encompasses' type.
Semantic check was successful.
Model : BANF fachlich freigeben
Rule: Only one single object of the 'Operation' type is connected with a system function.
Description: Only one single object of the 'Operation' type can be connected with a system function and interpreted via the transformation to the BPEL process.
Semantic check was successful.
Model : BANF fachlich freigeben
Rule: Only specific types of function symbols are used.
Description: Only the business function, the system function and the system function target are allowed for transformation to the BPEL process.
Semantic check was successful.

Figure A.7: Aris EPC rules (Rules for service-oriented EPC 1) [AG06]

Model : BANF fachlich freigeben
Rule: Process contains public messaging activities.
Description: Public messaging activities serve to specify the input and output information of the process. This information is used by other components when communicating with the process. According to the modeling conventions for service-oriented EPCs, public messaging activities can represent process steps that follow both start events and preceding end events and specify the associated input and output data objects of the process. Alternatively, the input and output data of the process can be specified as a data object of the start or end event.
Semantic check was successful.
Model : BANF fachlich freigeben
Rule: Process parallel flows, inclusive decision paths and exclusive decision paths are well-formed.
Description: Process parallel flows should be specified by splitting and joining AND/XOR rules, or they should contain either one splitting AND/XOR rule only for which there is no other connection between their paths, or one joining AND/XOR rule only that is met by all connections.
Semantic check was successful.

Figure A.8: Aris EPC rules (Rules for service-oriented EPC 2) [AG06]

Appendix C: CD Content

Appendix C describes the content of the attached CD.

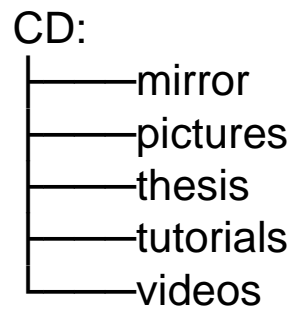


Figure A.9: CD Content

- Used Internet references were downloaded and saved in folder *mirror* on the CD.
- All figures of this thesis were saved in folder *pictures*.
- This work was saved in folder *thesis* on the CD.
- The following written tutorials explain proceedings that were relevant for the implementation of the prototype and can also be found on the CD in folder *tutorials*.
 1. **Testing a Web service on the SAP Discovery System:**
In this tutorial, the user logs on the SAP Discovery System, browses to a concrete Web service, runs a test with concrete parameters and gets a response. It is also explained where to check the effects of the concrete Web service.
 2. **Obtain WSDLs from the SAP Discovery System:**
The user learns how to find and obtain WSDLs on the SAP Discovery System.
 3. **Invoking Web services on the SAP Discovery System with WebSphere Integration Developer:**
In this tutorial, a process is created with Websphere Integration Developer where a concrete Web service on the SAP Discovery System is invoked. Also a description for testing the Web service call is given.
 4. **Invoking an Application on the SAP Discovery System via Java Connector (JCo) with WebSphere Integration Developer:**
This tutorial describes how an application on the SAP Discovery System can be invoked from WebSphere Integration Developer.

- Also two videos (“SOALiveSession_Part1” and “SOALiveSession_Part2”) can be found on the CD in folder *videos*. They demonstrate the steps that were shown in the SOA Live Session.