2009

MODEL-DRIVEN DEVELOPMENT OF SERVICEORIENTED BUSINESS APPLICATION SYSTEMS

Stefan Kätker
SAP AG, AP Engineering - Process Modeling

Susanne Patig
University of Bern, IWI

MODEL-DRIVEN DEVELOPMENT OF SERVICE-ORIENTED BUSINESS APPLICATION SYSTEMS

Stefan Kätker[1], Susanne Patig[2]

Abstract

The industrial development of real-world business applications that are based on service-oriented architecture (SOA) is subject to technological and organizational requirements. The paper describes an approach that tackles these requirements by ideas from model-driven development (MDD). The approach consists of a metamodel accompanied by modeling guidelines, a governance process ensuring that these guidelines are kept, and a modeling tool to support model creation and governance process. The successful application of the presented approach during the development of SAP's Business byDesign solution shows that the benefits of MDD in software industry lie beyond code generation and outweigh modeling effort.

1. Motivation

Application systems rely on *service-oriented architecture* (*SOA*) in three ways: 1) as a *composite* application for a particular purpose by integrating or extending existing SOA-based systems, 2) by being *a priori* and completely *designed* based on SOA principles or 3) by *ex post* providing services as external interfaces to their existing, *evolved* implementation. The second and third SOA usage types are perfect starting points for the first one. This paper introduces a methodology that supports the 'SOA by Design' development of application systems in a large, international project setting. The approach is called the *eSOA Modeling Methodology* and was developed by SAP AG.

To tackle the organizational and technological requirements of the development context (see Section 2), ideas from model-driven development were used. *Model-driven development (MDD)* is characterized by a tight coupling between the descriptions of a software system (*models*) and the system's implementation [7]: Software development starts from a set of models representing distinct views on software architecture [5]. These abstract models are successively transformed into platform-specific ones – with source code at the end. The benefits of applying MDD in software industry do not only stem from code generation, but also from the models that constitute an easy to understand lingua franca among the development teams [13].

The eSOA Modeling Methodology consists of a metamodel, a governance process and a tool environment; all of this is explained in Section 3 and related to the state of the art in Section 4. The approach has been applied in the development of the application system 'SAP Business byDesign' – SAP's ERP solution for small and medium enterprises, which was designed and implemented by

[1] SAP AG, AP Engineering - Process Modeling, Dietmar-Hopp-Allee 16, D-69190 Walldorf, Germany.
[2] University of Bern, IWI, Engehaldenstrasse 8, CH-3012 Bern, Switzerland.

strictly following SOA. The experience gained in this process is used to evaluate the methodology (Section 5); some critical points are shown.

2. Requirements of the Software Development Process

In the development process of the SAP ‚Business byDesign’ application system, the following three groups of requirements had to be considered:

1) *Large-scale commercial software development* often involves huge, decentralized teams. For example, more than 800 people from 3 continents participated in the development of SAP ‘Business byDesign’. Managing such a project requires clear boundaries between the development teams. Moreover, a common language for all developers is needed.
2) The development of a (functionally and technologically) new software system requires methodological support for the *whole software lifecycle* including requirements engineering, system design, system implementation, deployment and maintenance. Model-driven development satisfies these requirements, given that the models exactly match the implementation. This can be guaranteed by either source code generation form models or organizational means.
3) As a technological innovation, the new software system should be based on *service-oriented architecture*, which means that the software system provides its functionality by interacting (software[3]) services. *Services* reside on the implementation level, i.e., they are (software[2]) components that completely encapsulate the implementation of some functionality [16]. From the outside, only the description of this functionality (the *interface*) is visible. Interfaces group the operations offered by the service and the operation *signatures* (input and output parameters and their respective types). During the development of a SOA-based application system, the rather unstructured functional requirements of varying granularity must be broken down into well-defined and consistently structured components and interfaces.

These requirements are satisfied by the eSOA Modeling Methodology as follows:

1. *Metamodel* (see Section 3.1) *and content creation guidelines* (see Section 3.2)
   The metamodel contains all artifacts (metaclasses; see Section 3.1.1) to describe the service-oriented architecture of a (business) application system, i.e., the software components, their interfaces and their relationships to other components and to the system environment. The metamodel is accompanied by content creation guidelines for the identification, design and implementation of metaclass instances. The various views on software architecture [5] are realized by arranging the metaclasses into several model types (see Section 3.1.2).
2. *Governance process* (see Section 3.2):
   To ensure that the modeled content is consistent and complies with the guidelines, a central governance process is required. All models and entities have to pass a review before implementation. The governance process also covers changes during the development cycle. Software changes without subsequent model changes and governance approval are not allowed.
3. *Modeling tool environment* (see Section 3.3):
   Metamodel and content creation must be supported by a tool environment that implements all metaclasses and model types of the eSOA Modeling Methodology. The modeling tools create the eSOA repository for the modeled contents and include facilities to generate proxy code.

[3] We skip this term in the following - for brevity and since it is clear from the context.

3. eSOA Modeling Methodology

3.1. Metamodel

3.1.1. eSOA Metaclasses

SAP ‘Business byDesign’ relies on an enriched form of SOA called *Enterprise SOA* (*eSOA*, for short) [16], where the business functionality (*business scope*) of an application system is completely and solely exposed via services. The services are provided by business objects and process components.

Business objects are the metaclasses with the finest granularity; they structure the functionality of an application system. A *Business Object* (BO) is a set of entities with common characteristics and behavior that represents well-defined business semantics [12], [16]. The functionality of business objects can be accessed via services. Typical business objects are, e.g., ‚sales order’ and ‚customer invoice’ (see Section 3.1.2).

Each business object belongs to exactly one process component (see Fig. 1). Process components group business objects and are the building blocks for business processes. A *Process Component* (PC), for example, ‚Sales Order Processing’ (see Fig. 3 in Section 3.1.2), is a modular, context independent, reusable software package that exposes its functionality (which may span several database transactions) as (compound) services.

Process components are collected into *deployment units* (see Fig. 1). A Deployment Unit (DU) is a piece of software – like ‚Customer Relationship Management’ – that can be operated independently on a separate physical system. Usually, deployment units are jointly shipped. They are decoupled from each other via enterprise services and process agents (see below). Logical Units of Work are not created across deployment units.

The exposed services are abstractly described by service operations. A *service operation* is a specification of a function (e.g., ‚create sales order’) with a set of message types assigned as its signature. Semantically related service operations are combined into *service interfaces* (SI) that specify the functionality offered (inbound SI) or used (outbound SI) by process components (see Section 3.1.2, Fig. 3). Depending on the interacting partners, it is distinguished between A2A and B2B service interfaces, intended for the communication between applications of one business partner (A2A) or between systems at several business partners (B2B), respectively.

Process Agents (PA) are pieces of software responsible for the interaction between process components and across deployment units. They relate the sending (outbound PA) or receiving (inbound PA) of XML messages to changes of the involved business objects (e.g., status update, creation or deletion of BO instances) and, hence, intermediate between business objects and service interfaces (see Fig. 1). Fig. 3 in Section 3.1.2 provides an example for the use of process agents.

At the finest level, business objects are structured into (BO) nodes. A *business object node* represents a semantically related group of elements that can be treated as a part of the business object. For example, the business object ‘sales order’ may comprise the nodes ‘sales order item’, ‘price calculation’ and ‘delivery terms’. Business object nodes are typed by data types.

Fig. 1 shows the abstract syntax (i.e., the constructs and their allowed connections [8], [7]) of the eSOA metamodel metaclasses as an UML class diagram. This abstract syntax is accompanied by *eSOA Content Creation Guidelines* (see Section 3.2) for the identification, design and implementation of metaclass instances. The concrete syntax of the metaclasses, i.e., their notation [8], [7], is defined by the eSOA model types, which are described in the next section.

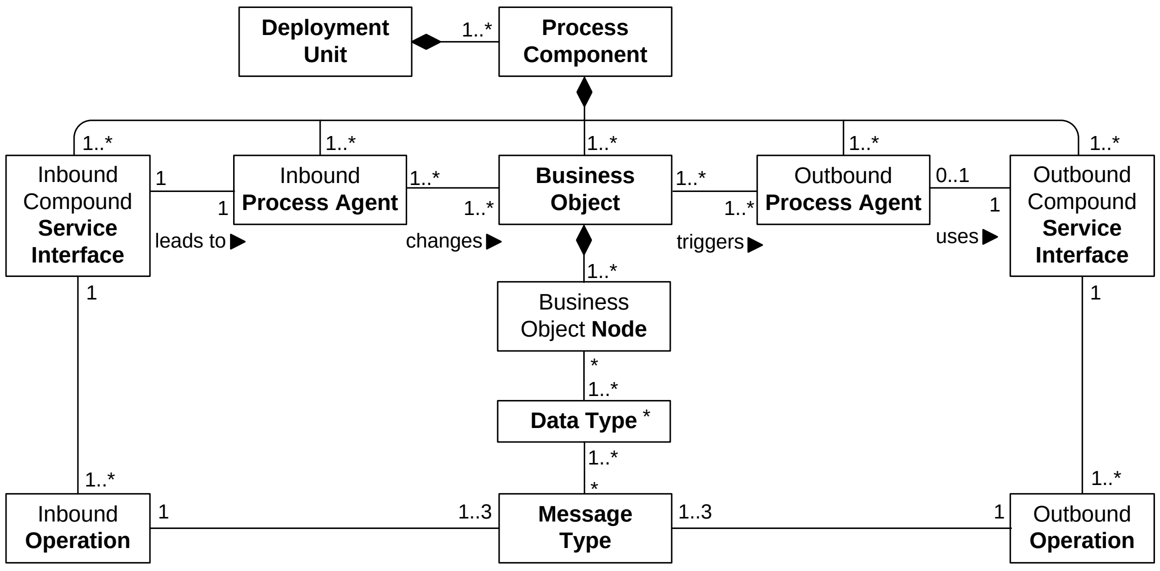


Fig. 1. Abstract Syntax of the Main eSOA Metamodel Artifacts.

3.1.2. eSOA Model Types

This section explains the main eSOA model types. According to ANSI/IEEE [2], they cover the structural (PC Models, BO Models), behavioral (Integration Scenario Models, PC Interaction Models) and context (Integration Scenario Models) viewpoints on software architecture. The model types are illustrated by the scenario „Sell from stock", which describes the fulfillment of a sales order for material from stock (Fig. 2 to Fig. 4).

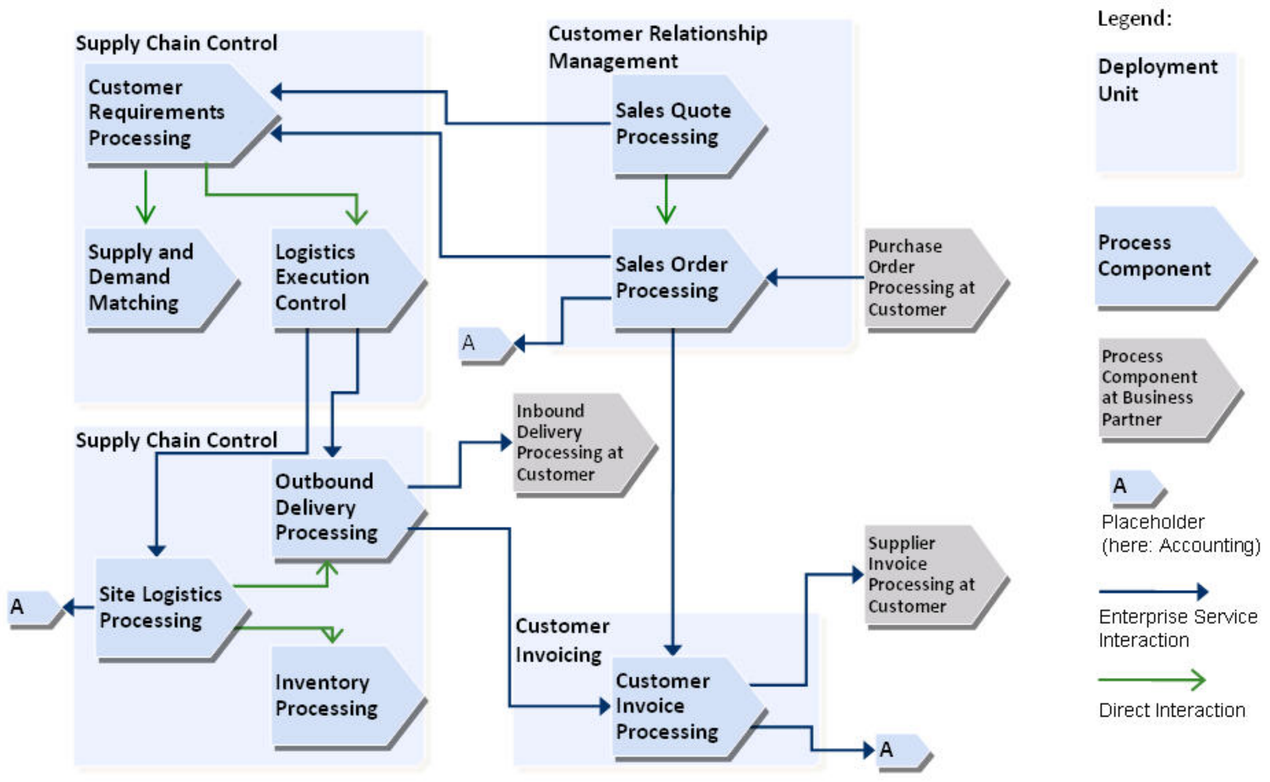

Supply Chain Control
Customer Requirements Processing
Supply and Demand Matching
Logistics Execution Control
Customer Relationship Management
Sales Quote Processing
Sales Order Processing
Purchase Order Processing at Customer
A
Inbound Delivery Processing at Customer
Supply Chain Control
Outbound Delivery Processing
Site Logistics Processing
Inventory Processing
A
Customer Invoicing
Customer Invoice Processing
Supplier Invoice Processing at Customer
A
Legend:
Deployment Unit
Process Component
Process Component at Business Partner
A
Placeholder (here: Accounting)
Enterprise Service Interaction
Direct Interaction

Fig. 2. Sample Integration Scenario Model for 'Sell from stock' (© SAP AG).

Integration Scenario Models (see Fig. 2) are the first ones created. They spring from the functional requirements and describe the realization of a business process by eSOA. An integration scenario consists of deployment units, process components and the necessary interactions among them. It is distinguished between *direct interactions* by local function calls, *service interactions* that amount to message exchange via the SAP Exchange Infrastructure (XI), and *other interactions* – executed by, e.g., remote function calls. All types of interactions are represented by arrows. Process components that do not belong to SAP 'Business byDesign' are also depicted. For example (see Fig. 2), the processing of sales orders is a typical task of customer relationship management and often the result of a sales quote. Each placed sales order corresponds to a purchase order at the customer. Thus, the DU ,Customer Relationship Management' consists of two PC (,Sales Quote Processing' and ,Sales Order Processing') that directly interact, whereas messages with the customer's systems must be exchanged via service interactions. Service interaction is also necessary to transfer the sales order's data to invoicing, as the PC ,Customer Invoice Processing' belongs to another DU.

Each PC is described by a *Process Component Model* (see Fig. 3). This model type shows the business objects and process agents belonging to some PC, its service interfaces and all process components using the inbound service interfaces. Process Component Models do not represent process flow logic, but architecture relevant 'uses' and 'realized by' relationships. Service interfaces placed above or below (to the right of) a BO group asynchronous (synchronous) interactions.

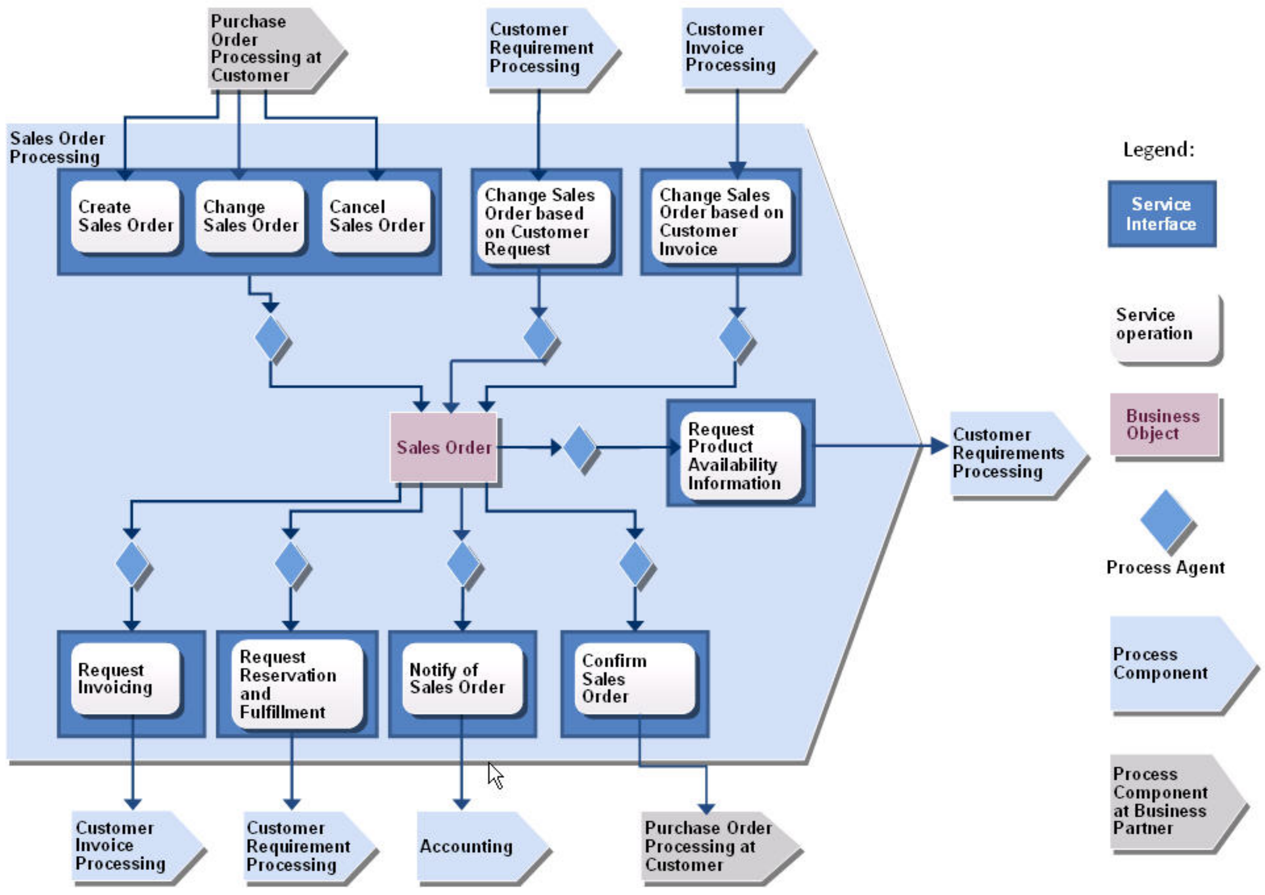

Purchase Order Processing at Customer
Customer Requirement Processing
Customer Invoice Processing
Sales Order Processing
Create Sales Order
Change Sales Order
Cancel Sales Order
Change Sales Order based on Customer Request
Change Sales Order based on Customer Invoice
Sales Order
Request Product Availability Information
Customer Requirements Processing
Request Invoicing
Request Reservation and Fulfillment
Notify of Sales Order
Confirm Sales Order
Customer Invoice Processing
Customer Requirement Processing
Accounting
Purchase Order Processing at Customer
Legend:
Service Interface
Service operation
Business Object
Process Agent
Process Component
Process Component at Business Partner

Fig. 3. Sample Process Component Model (© SAP AG).

Fig. 3 shows the Process Component Model of the PC ,Sales Order Processing' from Fig. 2. The operations to create, change or cancel instances of the BO ,Sales Order' are grouped into the same service interface since they can be asynchronously invoked by the same PC ‚Purchase Order Processing at Customer'. A process agent (a technical artifact) connects the SI to the BO.

The *Process Component Interaction Model* (see Fig. 4) can be seen as an extended view on the information contained in Process Component Models: It shows the service choreography and the messages exchanged between exactly two process components for a specific business goal and the involved business objects, process agents, interfaces, operations and message types. This model type is only generated for service interactions. Fig. 4 shows the corresponding service interaction between the PC ‚Customer Invoice Processing' and the PC ‚Sales Order Processing', which exchange special message types for customer invoices (request and issued confirmation).

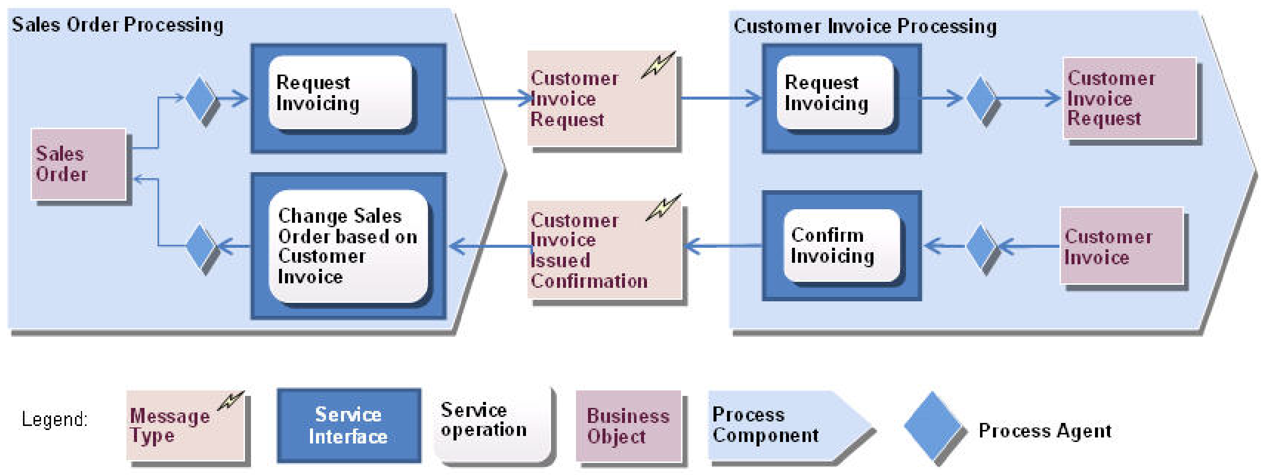


Fig. 4. Sample Process Component Interaction Model (© SAP AG).

3.2. Content Creation and Governance Process

There is no central 'modeling department' within SAP AG; instead, the modeling (called *content creation* in the following) is a task of the developers. Decentralized content creation is facilitated by the *eSOA Content Creation Guidelines* that accompany the eSOA metamodel as well as centralized methodology coaching: The subject matter experts in the development teams are assisted by eSOA Modeling Methodology experts, who help them in applying the content creation guidelines.

The content creation guidelines control the identification, design and implementation of metaclass instances and models; a few examples are given in the following: The *identification* of metaclass instances puts the eSOA metaclass definitions into particular content entities. For example, business objects are defined according to industry standards, e.g., the ones for B2B communication from RosettaNet [11], or industry best practices. The formation of process components follows an outside-in business view: In general, a process component represents the software realizing a business process that is typically executed in one department. A deployment unit is a collection of process components and corresponds to the smallest unit of software distribution. Deployment units are determined by thinking about customer's deployment scenarios (e.g., central accounting, line-of-balance-oriented production and regionally distributed sales) and business process outsourcing. Service interactions and the message types exchanged are organized in patterns (e.g. ‚notification', ‚information distribution', ‚query response'), which are strictly derived from the transaction patterns of the UN/CEFACT modeling methodology [14].

Design issues covered by the content creation guidelines include, for example, naming rules, the determination of identifiers, modeling restrictions (e.g., a BO node data type must be either a global data type or a data type composed out of global and aggregated data types) and technical restrictions (e.g., multi-valued attributes are not allowed for BO nodes). The content creation guidelines also provide code skeletons for the *implementation* of the metaclasses.

Finally, the *PIC (Process Integration Content) governance process* is SAP's central approval and quality assurance procedure for the modeled eSOA contents. Its tasks consist in 1) checking the conformance of the models to the eSOA content creation guidelines, 2) approving exceptions from these guidelines, 3) assuring identical semantics as well as reuse of the modeled entities across development teams and throughout SAP 'Business byDesign' and 4) managing model changes. Thus, the process increases the consistency of the modeled contents.

In detail, the PIC governance process comprises several phases (see Fig. 5). PIC0 focuses on the definition of process integration aspects, i.e., the service orchestration between components: First, the functional requirements are structured into business objects, process components and deployment units by applying the eSOA content creation guidelines. PIC01 approves cut, semantics (i.e., appropriate definition) and naming of the identified entities. Then, the constituents of the Integration Scenarios Models (process components and their interactions) are defined and confirmed by PIC02. Thirdly, each interaction between process components is detailed by Process Component Models and Process Component Interaction Models (PIC03). The phases PIC1 to PIC3 are concerned with data modeling: Business objects, their nodes and data types as well as service signatures are defined. Fig. 5 relates the PIC governance process to the process of modeling.

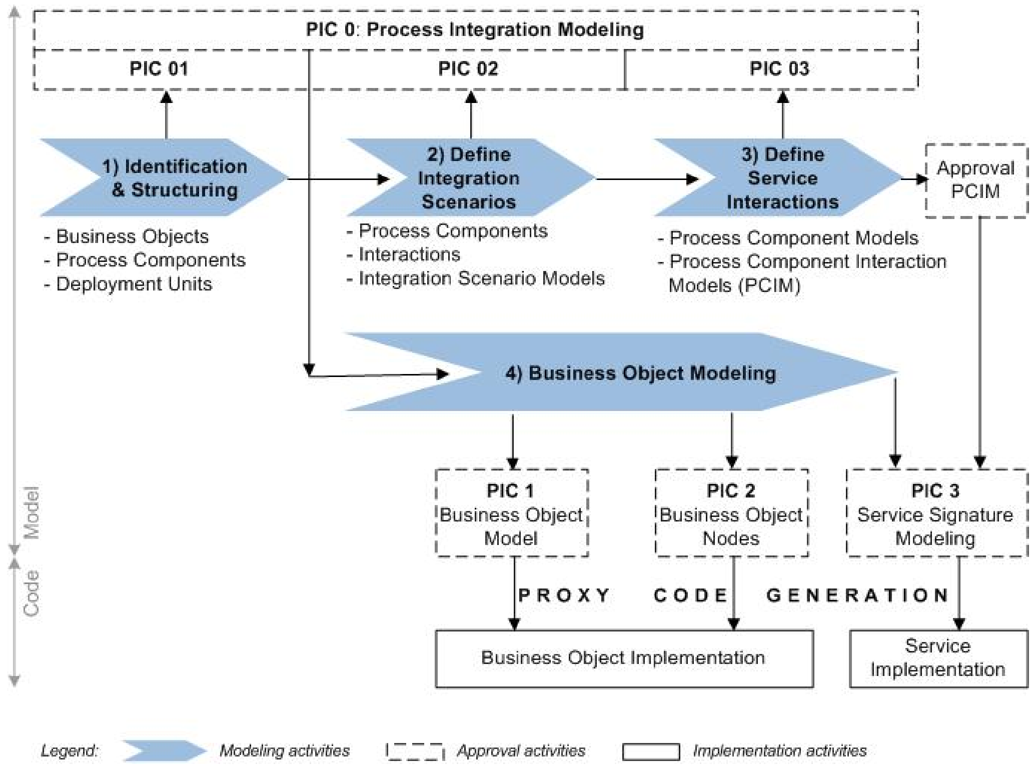

PIC 0: Process Integration Modeling
PIC 01
PIC 02
PIC 03
1) Identification & Structuring
2) Define Integration Scenarios
3) Define Service Interactions
Approval PCIM
- Business Objects
- Process Components
- Deployment Units
- Process Components
- Interactions
- Integration Scenario Models
- Process Component Models
- Process Component Interaction Models (PCIM)
4) Business Object Modeling
PIC 1 Business Object Model
PIC 2 Business Object Nodes
PIC 3 Service Signature Modeling
PROXY CODE GENERATION
Business Object Implementation
Service Implementation
Model
Code
Legend:
Modeling activities
Approval activities
Implementation activities

Fig. 5. SAP Process Integration Content (PIC) Governance Process.

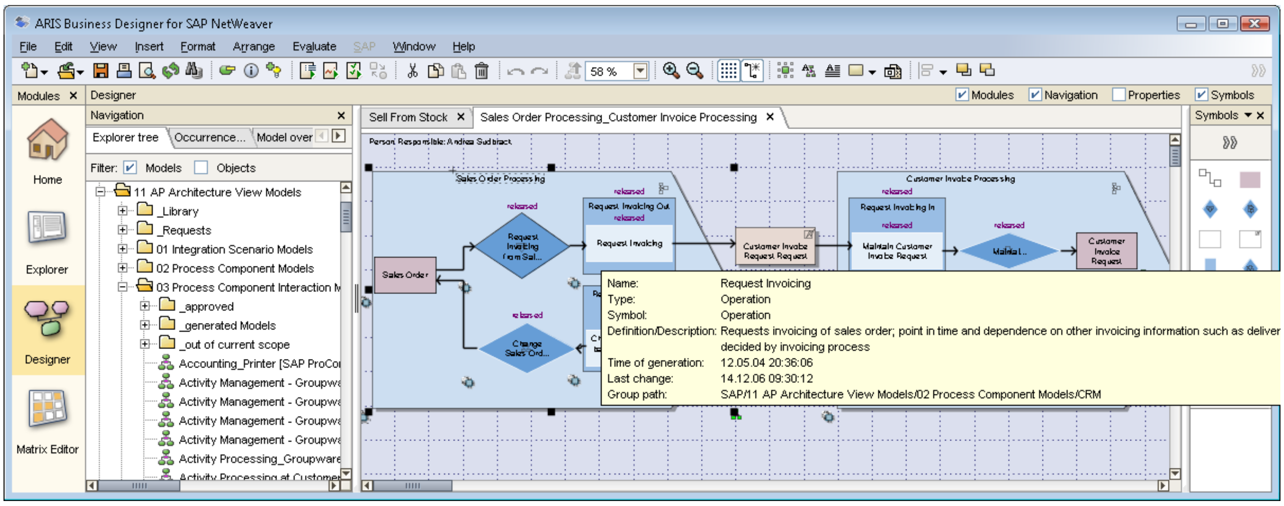
Only those modeled entities that have passed the PIC governance process are released for implementation and listed in the *eSOA Repository (ESR)*, a central directory of all elements that embody eSOA [16]. For business objects and service interfaces, proxy coding is generated (see Fig. 5).

If either subsequent functional requirements or unforeseen implementation issues call for deviations from the modeled entities, first, the affected models have to be adjusted. Then, the PIC governance process must be passed anew (now as a change process), before implementation can be started.

3.3. Modeling Tool Environment

To efficiently realize MDD in large software development projects, an appropriate tool environment is important. Not only modeling must be supported, but also the governance process. Therefore, key features of such a tool environment include: Close integration with the eSOA repository, high usability and specific adaptation to the modeling methodology to make modeling efficient, status management for the models and their entities to track the approval status in the governance process, versioning and release management like for source code. Reports should check automatically if abstract syntax and content creation guidelines have been kept. These checks ensure consistency and quality of the modeled contents.

Owing to the development cooperation between SAP and IDS Scheer, the ARIS tool [1] was used to implement the eSOA Modeling Methodology. The close integration with SAP's eSOA Repository as well as the flexible reporting and configuration capabilities of ARIS proved to be very useful. However, ARIS provides only the tool basis to implement the eSOA Modeling Methodology (see Fig. 6); the ARIS modeling methodology featuring event-driven process chains is not a part of the eSOA Modeling Methodology.


ARIS Business Designer for SAP NetWeaver
File Edit View Insert Format Arrange Evaluate SAP Window Help
Modules Designer Navigation Explorer tree Occurrence... Model over
Filter: Models Objects
11 AP Architecture View Models
_Library
_Requests
01 Integration Scenario Models
02 Process Component Models
03 Process Component Interaction M
_approved
_generated Models
_out of current scope
Accounting_Printer [SAP ProCo
Activity Management - Groupw
Activity Processing_Groupware
Home
Explorer
Designer
Matrix Editor
Sell From Stock
Sales Order Processing_Customer Invoice Processing
Sales Order Processing
Customer Invoice Processing
released
Request Invoicing Out
Request Invoicing
Customer Invoice Request Request
Request Invoicing In
Maintain Customer Invoice Request
Customer Invoice Request
Sales Order
Change Sales Ord...
Symbols

Name: Request Invoicing
Type: Operation
Symbol: Operation
Definition/Description: Requests invoicing of sales order; point in time and dependence on other invoicing information such as deliver
decided by invoicing process
Time of generation: 12.05.04 20:36:06
Last change: 14.12.06 09:30:12
Group path: SAP/11 AP Architecture View Models/02 Process Component Models/CRM

Fig 6. A Process Component Interaction Model in the ARIS Modeling Tool.

4. Relationship of the eSOA Metamodel to Standards

Although the eSOA Modeling Methodology was invented to support the development of a company-specific form of SOA, the resulting service-oriented application systems will probably be integrated in landscapes of heterogeneous software systems. For that reason, conformance to standards is important. Especially standards for the description of software architectures (ANSI/ IEEE 1471-2000 [2]), general modeling standards (UML [8]) and standards related to (web) services (WSDL [15], SCA [10]) must be kept. Process modeling standards like the Business Process Modeling Notation (BPMN [6]), event-driven process chains (EPC [1]) or UML activity diagrams [8] are, however, not relevant, since software architecture, which the eSOA Modeling Methodology describes, is only concerned with structural aspects of software systems and general interaction possibilities (as opposed to particular interactions) [5].

In fact, the eSOA metamodel can largely be expressed as an UML profile. By accepting loss of some internal information, eSOA models can also be transformed into a SCA-compliant representation. More detailed descriptions of these mappings can be found in [4]. The main advantage of our approach is the precise mapping from the eSOA models to SAP's application architecture.

The eSOA definitions of service interfaces and operations agree with the standard *Web Service Description Language* (WSDL) 2.0. This conformance is relevant to external invocations of the services provided by SAP 'Business byDesign'. WSDL 2.0 characterizes services by interfaces that contain at least one operation; the signature of an operation consists of messages [15]. The transformation from WSDL to proxy code is straightforward.

5. Evaluation of the eSOA Modeling Methodology

The eSOA Modeling Methodology was applied during the development of the service-oriented application system SAP 'Business byDesign': The functional requirements were decomposed into business objects and services and aggregated to process components and deployment units before actual coding started. Table 1 shows the according numbers of eSOA entities.

Table 1. Numbers and Usage of Models (March 2008)

Model Type		**Model Constituents**						**Model Usage beyond Documentation**
		DU	PC	BO	PA	O	MT	
Name	**Counts**	**20**	**130**	**373**	**622**	**651**	**579**	
Integration Scenario Models	**28**	X	X					Derivation of end-user test cases
PC Models	**115**			X	X	X		Generation of PC Interaction Models, Consistency Checks
PC Interaction Models	**211**		X	X	X	X	X	Activation of a service interaction during configuration generates entries in SAP XI
BO Models	**373**			X				Derivation of operation signatures; Generation of provider classes, interfaces and proxies

Abbreviations: BO: Business object, DU: Deployment unit, MT: Message type, O: Operation, PA: Process agent, PC: Process component, X: Model type uses metaclass instances

The creation of the models causes effort. The effort shows up in the numbers of models and their entities (see Table 1) and in the phases of the governance process (see Fig. 5) that must be passed before implementation and in case of changes. Although this effort is significant, our experience is that it pays off through the usage of the models over the whole *software lifecycle* (see Table 1): The eSOA metamodel provides clear guidance and a common language to formalize the gathered functional *requirements* in a way that immediately leads to a service-oriented system *design*. System *specification* is structured according to the eSOA metaclasses and model types (for example, specifications are written per process component or integration scenario); *development teams* are formed around process components and deployment units. In *implementing* the system, provider classes, interfaces and proxies have been automatically generated from the Business Object Models in the eSOA Repository. End-user *test* cases are derived from Integration Scenario Models; tests related to data and message types can also be generated. In order to derive the technical configuration of a particular customer system during *deployment*, Process Component Integration Models are used to automatically create interaction-specific configuration entries for enterprise service interactions in SAP's Exchange Infrastructure (XI). Software *maintenance* requires deep knowledge of the system and its integration; the service and support teams gain this knowledge form the modeled content. Finally, the common language coined by the eSOA metamodel drives development efficiency throughout the entire software lifecycle.

Two critical points of the eSOA Modeling Methodology can be identified; the first one is the governance process. Though it is mandatory for quality assurance, it should not be overstated. Governance must balance out rigid quality checks and room for innovation and exceptions. The PIC governance process was refined several times to ensure that it does not become the bottleneck of development efficiency, but still guarantees consistent content of high quality.

Secondly, the eSOA Modeling Methodology is architecture-driven. The created models are not well suited to explain business experts how business processes are realized. Experience showed that even in developing composite applications more business context is required to manage service integration. This can be achieved by combining service interaction and process flow as proposed by BPMN and the related execution language BPEL. Therefore, our future research is directed to extending the eSOA Modeling Methodology by a BPMN-based process flow description.

6. Conclusion

This paper has presented an approach for the model-driven development of SOA-based business application systems in a huge industrial setting. In domains other than business applications, both metamodel and content creation guidelines of the eSOA Modeling Methodology must be adapted.

The eSOA Modeling Methodology belongs to the development paradigm of SAP's 'Business by-Design' application system. As a result, the complete functional content of 'Business byDesign' is described by models with a one-to-one correspondence to source code. The models form an easy to understand common language across various functional and organizational areas. In a project involving more than 800 developers, such a language raises development productivity

The modeled content opens up opportunities to design and document other aspects of the software system as well, such as configuration constraints, process flow and end-user interfaces. All this can only be realized if metamodel, content creation guidelines, governance process and modeling tool are treated as a single, integrated topic, as done by the eSOA Modeling Methodology.

7. References

[1] IDS Scheer, ARIS Software, http://www.ids-scheer.com/en/aris.

[2] IEEE, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Std 1471-2000. Recognized as an American National Standard (ANSI), Approved 2000-09-21.

[3] JOHNSTON, S.K., UML 2.0 Profile for Software Services, http://www.ibm.com/ developerworks/

[4] KÄTKER, S., PATIG, S., Model-Driven Development for Service-Oriented Software Systems. Preprint No. 208, Institut für Wirtschaftsinformatik der Universität Bern, Bern, 2008.

[5] KRUCHTEN, P.B., The 4+1 View Model of Architecture, in: IEEE Software Vol. 12, No. 6 (1995).

[6] OMG, Business Process Modeling Notation, Version 1.1, formal/2008-01-17 http://www.bpmn.org/

[7] OMG, Model-driven Architecture, Guide Version 1.0.1, omg/2003-06-01, http://www.omg.org/

[8] OMG, Unified Modeling Language: Superstructure, Version 2.1.2, formal/2007-11-02, http://www.omg.org/

[10] OSOA, Service Component Architecture Assembly Model V1.00, http://www.osoa.org/

[11] RosettaNet: http://www.rosettanet.org/

[12] SEUBERT, M., Business Objects und objektorientiertes Prozeßdesign (in German), in: Becker, J., Rosemann, M., Schütte, R. (Hrsg.), Entwicklungsstand und Entwicklungsperspektiven der Referenzmodellierung, S. 46-64. Arbeitsbericht Nr. 52, Universität Münster, Institut für Wirtschaftsinformatik 1997.

[13] STARON, M., Adopting Model Driven Software Development in Industry – A Case Study at Two Companies, in: Nierstrasz, O. et al. (Hrsg.), MoDELS 2006, LNCS Vol. 4199, pp. 57 – 72. Springer, Berlin 2006.

[14] UN/CEFACT, UMM User Guide; http://www.unece.org/cefact/umm/umm_index.htm.

[15] W3C, Web Services Description Language (WSDL) Version 2.0, Part 1: Core Language. http://www.w3.org/

[16] WOODS, D., MATTERN, T., Enterprise SOA – Design IT for Business Innovation, O'Reilly, Beijing 2006.