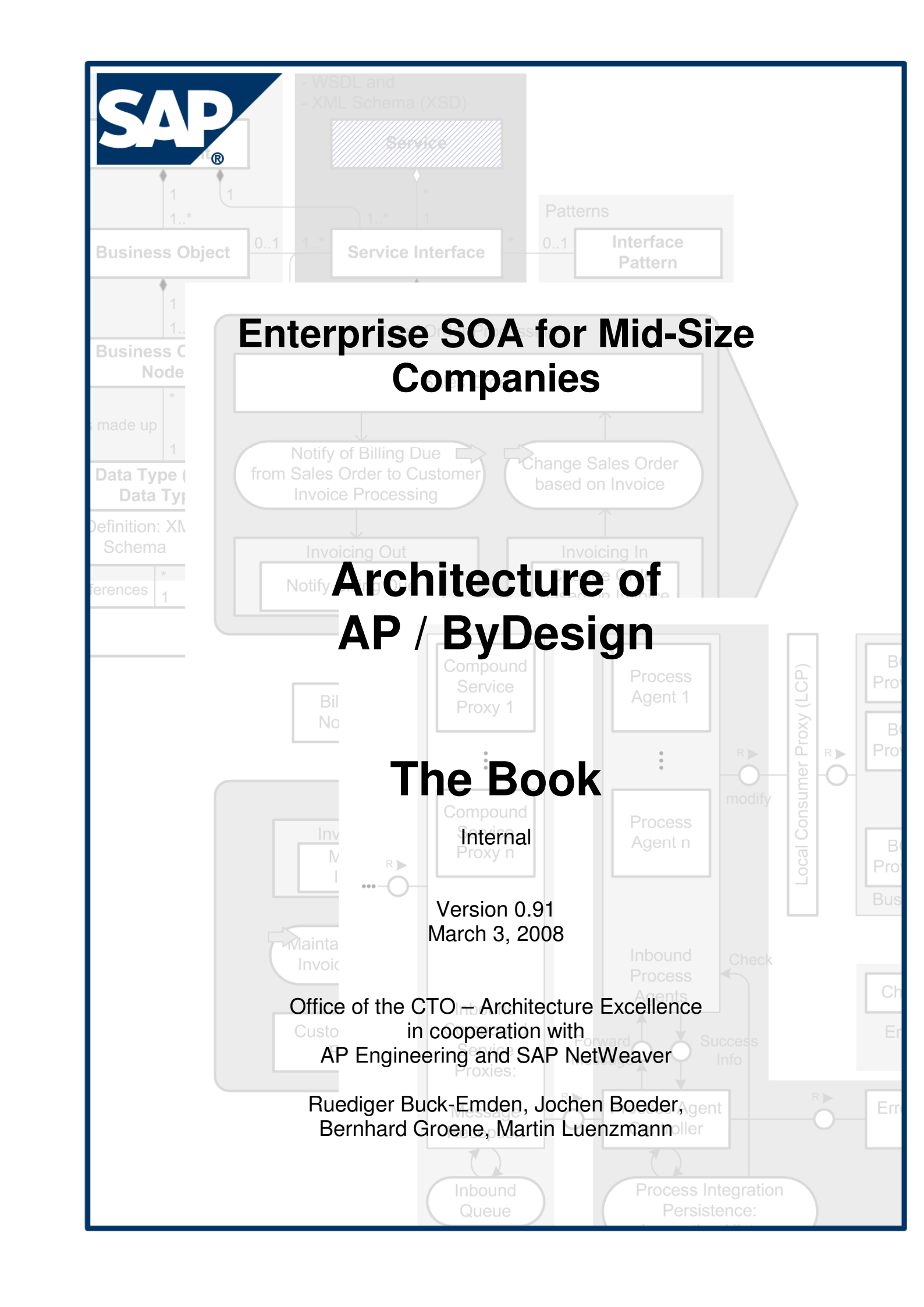


Enterprise SOA for Mid-Size Companies

Architecture of AP / ByDesign

The Book

Internal

Version 0.91
March 3, 2008

Office of the CTO – Architecture Excellence
in cooperation with
AP Engineering and SAP NetWeaver

Ruediger Buck-Emden, Jochen Boeder,
Bernhard Groene, Martin Luenzmann

Table of Contents

Table of Figures

Foreword

Enterprises have an ongoing demand for software solutions that give them the flexibility to react immediately to changing business requirements. With SAP Business ByDesign, midsize companies now have a business software product at hand that enables them to efficiently run, adapt, and grow their software landscape as needed to optimize their business.

This book is about the technical architecture and concepts behind SAP Business ByDesign. These are based on the enterprise service-oriented architecture (enterprise SOA), SAP's blueprint for adopting the service-oriented paradigm in the context of business computing. Of course, this book is very much about service provisioning and service consumption, but it is also about business process integration, user interface design, analytical reporting, configuration, extensibility, and system operation. And it is about model-driven development, a key feature of enterprise SOA that enables higher flexibility and lower total cost of ownership across the entire software life cycle.

Many colleagues contributed with valuable input, thoughts, discussions, and feedback. Without their support, we would never have been able to finalize this book. The following people deserve our special thanks:

Werner Aigner, Stefan Baeuerle, Peter Barker, Henning Blohm, Carsten Boennen, Rainer Brendle, Cristina Buchholz, Josef Dietl, Peter Eberlein, Tahar El Idrissi-Lamghari, Thomas Fiedler, Robert Getzner, Stefan Girsig, Jochen Guertler, Reiner Hammerich, Jan Heiler, Manfred Hirsch, Stefan Kaetker, Klaus-Peter Lang, Peter Lorenz, Brenda MacKay, Olaf Meincke, Holger Meinert, Gordon Muehl, Arnold Niedermaier, Guenter Pecht-Seibert, Michael Pohlmann, Jochen Puzicha, Thomas Rinneberg, Bare Said, Thomas Schneider, Uwe Schulz, Pradeep Kumar Singh, Vishal Sikka, Axel Uhl, Frank Wagner, Walter Weber, Volker Wiechers, Georg Wilhelm, and Sheila Zelinger.

Rüdiger Buck-Emden

1 The Business Challenge for Midsize Companies

Midsize companies operate in many diverse businesses and vary from industry to industry as well as by country and region. Despite this diversity, many of them suffer from the same challenges, which center around improving business performance at predictable and affordable costs. Also important is the fact that the business processes of midsize companies are not very different from those of larger enterprises. Overall, midsize companies share the same objectives as large enterprises, but need to reach these goals with fewer resources in a shorter period of time.

In a press release, Gartner Inc. reports that only one-third of information technology (IT) spending in midsize companies directly improves business performance, while nearly two-thirds is spent on IT infrastructure like networks and servers and on utility applications such as e-mail and payroll (see [Gart05]). But this spending on IT infrastructure will not be sufficient to ensure sustainable, profitable growth in a competitive, globalized economy. Rather, successful mid-market companies have to think about how to retain their flexibility to quickly respond to new customer and market needs, how to optimize their operations, and how to maintain their deep level of customer intimacy in the face of growth. This means that striving for lower total cost of ownership (TCO) alone cannot serve as a promising long-term strategy. In addition, any company intending to be a leader in products, services, and operations must learn to evaluate and enhance their business performance continuously.

To achieve this goal, a growing number of midsize companies is looking for software solutions that provide the flexibility they need to react immediately to changing business requirements. They need software solutions that enable them to gain competitive advantage by supporting continous adaptability of business processes, organizational structures etc., and at the same time allow for enhanced productivity and predictable cost of ownership.

SAP addresses this demand with SAP Business ByDesign, a business software that allows midsize companies to efficiently run, adapt, and grow their software landscape as needed to optimize their business. This ability to enable continuous change at predict-

able cost stems from the service-oriented architecture of SAP Business ByDesign, with the SAP business process platform in the center, a rich technical infrastructure (technology platform) accompanied by reusable software building blocks (application platform).

SAP Business ByDesign is intended to provide functionality for various business areas like financials, human resources, customer relationship management, and supply chain management. This business functionality is not captured in individual application silos, but is available as building blocks to be be reused in various business processes according to customer-specific needs. It is this combination of reusable business software components and matching technical infrastructure that creates the flexibility midsize companies are looking for.

This book does not go into the details of the business functionality of SAP Business ByDesign. However, it explains to the software developer community the details of the technical architecture, which has been designed to drive down TCO, increase ease of use, and allow affordable agility and flexibility of business processes for midsize companies.

2 Enterprise Service-Oriented Architecture

for SAP Business ByDesign from the Architect's Point of View

People welcome service-oriented architecture (SOA) as a great architecture approach to address the challenges of improving business performance and lowering TCO. From the general perspective, there is a broad agreement of what a service-oriented architecture could be. But in detail, people tend to have various perceptions. Many authors addressed this topic from different angles, resulting in slightly diverse definitions of the term SOA. Examples are:

"A service-oriented architecture is a framework for integrating business processes and supporting IT infrastructure as secure, standardized components – services – that can be reused and combined to address changing business priorities." (See [BBF+05], page 5.)

"A Service Oriented Architecture (SOA) 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." (See [KBS05], page 57.)

Many of the popular definitions identify concepts such as loose coupling, service contracts, autonomy, abstraction, reusability, composability, statelessness, and discoverability as key aspects of a service-oriented architecture (see for instance [ERL05], page 37). In general we agree with this characterization of SOA, but do not want to repeat the details of this discussion here. Instead we like to refer to a definition from a paper published by three IBM architects in 2004, a definition that fits very well with our thinking and the subject of this book.

"SOA is the architectural style that supports loosely coupled services to enable business flexibility in an interoperable, technology-agnostic manner. SOA consists of a composite set of business-aligned services that support a flexible and dynamically re-configurable end-to-end business processes realization using interface-

based service descriptions. … Individual or collections of services that enjoy various levels of granularity can be combined and choreographed to produce ‘new’ composite services that not only introduce new levels of reuse but also allow the dynamic reconfiguration of business systems.” (See [ABH04].)

This definition corresponds perfectly with what SAP refers to as enterprise SOA, meaning the adoption of the SOA approach in the context of enterprise business computing. SAP has taken the idea of SOA, has merged that with its technology platform SAP NetWeaver, and has combined that with a repository of services. Enterprise SOA provides the architecture blueprint for all new SAP offerings. This is especially true for SAP’s new solution for midsize enterprises, SAP Business ByDesign, whose architecture is discussed throuhout this book.

2.1 Getting Started

Isolated and inflexible business applications addressing only one specific functional area like customer or supplier management are widespread, but they are loosing ground. Companies are turning away from closed business application silos with all their restrictions and instead are looking for affordable software solutions that can be adapted quickly and easily to ever-changing business processes across the whole company and beyond. What we see now are software suites of reusable building blocks and configurable end-to-end processes that provide the best support and the highest degree of flexibility to run the business.

The basic idea behind SAP’s approach to enterprise SOA is to build business applications from independent, reusable building blocks that communicate via standardized Web service interfaces. What from the end-user perspective still looks like a traditional application now becomes more a façade to a demand-driven orchestration of business processes. This approach gives companies the flexibility they need to optimize their business while lowering the cost of implementation and operation.

Despite this enhanced concept for creating business applications, we have to concede that enterprise SOA is not completely new. It combines and extends a number of existing concepts with proven value for companies and business software solutions. In particular, enterprise SOA is influenced by the following:

- Business process orientation, moving enterprises away from the traditional focus on functional departments (and accompanying software solutions) towards end-to-end optimization of their business processes (within a company and beyond)
- Abstraction through modeling on various levels, resulting in lower cost of software development and support, as well as easier adaptation to customer-specific needs
- Client-server computing, leading to clearly distinguished software components acting as service providers, service consumers, or both
- Object orientation, resulting in a strict encapsulation of data and functionality (among other aspects)
- Loose coupling of software components, enabling flexible adaptation of business processes
- Event-driven architectures, allowing loose coupling of software components based on event-triggered asynchronous message communication
- Opennes through Web standards like eXtensible Markup Language (XML), Web Service Definition Language (WSDL), Universal Description, Discovery and Integration (UDDI), and SOAP, as well as industry-driven business process and communication standards like RosettaNet (high tech), CDIX (chemicals), and PDIX (petroleum)
- Platform-based development approach, supporting reuse of standardized components similar to that in other industries such as automotive manufacturing

All these concepts have been around for a while. But now it is time to take the best out of each, combine them in the right way, find the right levels of abstraction and granularity, identify the building blocks suitable for business, and take advantage of the prevalent agreement in the industry on widespread Web standards.

How all these fundamental concepts found their way into enterprise SOA is described in the following chapters.

2.2 Service-Oriented Architecture

As a prerequisite for diving into the details of enterprise SOA, let's have a quick glance at SOA in general. The basic idea of SOA is

to divide software into loosely coupled components that interact via well-defined service interfaces and that allow the composition of new applications from existing software components. With this in mind, we first want to introduce the following key concepts of SOA: services, Web standards, and composite applications.

2.2.1 Services

A service is a callable runtime resource that performs a coherent set of tasks upon request by a consumer. In a scenario where software components communicate via services, one component acts as the service consumer sending a service request to another component, the service provider. The service provider processes the request and sends the result as a service response back to the service consumer (see figure 2-1). This communication can be performed in a synchronous or asynchronous fashion (see chapter 4.1.3).

Services can be called only via their service interface. The service interface is self-contained and abstracts completely from the implementation of the functionality and the underlying technical platform. To facilitate reuse, interface design has to follow common rules, with Web standards being most popular.

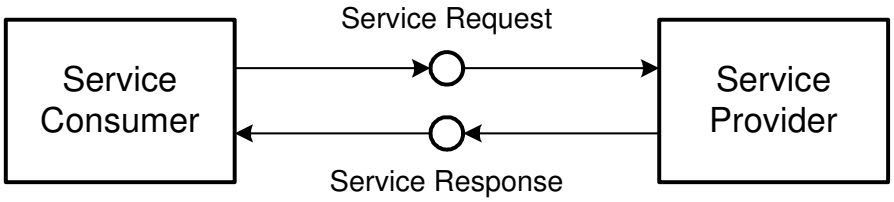


Figure 2-1 Loosely Coupled Software Components Interacting via Services

2.2.2 Web Standards

In principle, a service-oriented architecture can be implemented with any service-enabling technology like Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), or Enterprise JavaBeans (EJB). But without additional abstraction and standardization, service interaction remains limited to a certain technology environment. If, in contrast, services from different systems could communicate in a technology-agnos-

tic fashion, interaction would be much simpler. This is the target of Web service standards like WSDL, XML, SOAP, and UDDI.

WSDL is the standard issued by World Wide Web Consortium (W3C) for the definition of Web services. The importance of Web services is not that we now are able to interact via a service, which actually is not a new concept, but that Web services represent a generally agreed-upon standard. With Web services, different applications from different software vendors start speaking the same language – at least on the technical level. When calling Web services, XML messages are exchanged via SOAP typically using Hypertext Transfer Protocol (HTTP) for transportation.

UDDI is a platform-independent, XML-based registry enabling businesses to quickly and easily find and use Web services over the Internet. To identify a desired service, the service consumer interrogates the registry by SOAP messages. UDDI provides access to WSDL documents describing the protocol bindings and message formats required to interact with selected Web services as well as to additional metadata needed for service consumption.

Besides the technical aspect, the semantics of service interfaces is harmonized through additional business data standards like UN/CEFACT Core Components Technical Specification (CCTS), which provides a conceptual language and methodology for the identification of reusable information items called core components. Thanks to common naming rules and common core data types, service interfaces that offer similar or the same functionality look similar. In addition, mapping between data types can occur automatically. On top of these business data standards, the data exchange flow between business partners is governed by business process and communication standards like RosettaNet (see chapter 2.1).

With broad consent to these Web standards, the business software community has a great opportunity to build business software solutions that are much more open and flexible than they were in the past. This can, for instance, mean a call to a single service advertised by somebody else via UDDI, or it can mean the building of a new application out of reusable services to supply a specific need. The latter case is discussed in the following section.

2.2.3 Composite Applications

The availability of well-defined services leads directly to the concept of composite applications (see [BB03]). On a very general level, a composite application can be understood as an application that is created by combining multiple services according to the rules of a service-oriented architecture. Services for composite applications can be provided by software components or by other applications or systems.

Although this general definition is not wrong, it does not really reveal how well-designed composite applications differ from traditional business applications. But understanding this difference is important to realizing the opportunity companies have with composite applications:

Traditional business applications were built to collect and analyze business data and to automate the processing of these data. They could do this very efficiently but they fell short of integrating the user with user-centric collaborative and interactive processes and did not encourage the establishment of novel or flexible business processes. And, last but not least, they were self-contained and restricted by the assumption that they include and control all data and business logic needed. Now, properly designed composite applications can address these challenges with the following features, which go beyond the fact that they just combine multiple services:

- Provide a unified intelligent user experience across system boundaries (which also includes bridging the gap between business functionality and user productivity tools like office applications)
- Transcend functional, system, and even company boundaries and view business processes as a whole rather than its separate parts
- Are independent from the underlying service provider's implementation and life cycle

With enterprise SOA, SAP and its partners are able to extend SAP standard products by building composite applications according to the individual needs of their customers. If productized by SAP, these composite applications are called SAP® xApps™ composite applications. For a more technical discussion of composite applications, see chapter 8.3.

2.3 Enterprise SOA for AP / ByDesign

Now, with a basic understanding of SOA in place we can move forward to enterprise SOA, which is SAP's blueprint for a Web-based service-oriented architecture. This blueprint emphasizes the specific requirements of business solutions, thereby providing the foundation for service-enabling all SAP product lines for large, midsize, and small companies: SAP Business Suite, SAP Business ByDesign, and SAP Business One, respectively. See also [WoMa06] for a comprehensive discussion of enterprise SOA.

Keeping this in mind, we keep the focus of this book on describing how enterprise SOA is implemented within SAP Business ByDesign. The key elements to be discussed first are enterprise services, business objects, process components, and the business process platform (BPP).

2.3.1 Enterprise Services

From an availability point of view, services can be distinguished in two categories: those provided for internal use only and those offered to external parties as well. The latter are called enterprise services and have the following characteristics:

- Are based on Web service standards
- Provide specific business functionality
- Are well documented
- Afford guaranteed quality and stability

Enterprise services are created according to the predefined, harmonized enterprise SOA metamodel as described in chapter 3.3.1. The design-time environment for enterprise services is the enterprise services repository (ES Repository). Application developers use ES Repository to define the enterprise service interfaces, to model the respective business objects, and to specify the message choreography between service consumer and provider. The meta-level models and definitions in ES Repository are used to generate programming language-specific representations later on.

Besides ES Repository, two additional enterprise services catalogs serve specific purposes. An inventory of all available enterprise services together with accompanying documentation and possible usage scenarios is published in the enterprise services workplace of SAP Developer Network (www.sdn.sap.com). And for configur-

ing, publishing, and discovering enterprise services in the context of a specific customer installation, companies can use the SAP UDDI-based enterprise services registry, which is part of every customer system landscape.

2.3.2 Business Objects as Service Providers

In AP / ByDesign business objects are the principal service providers. Business objects encapsulate semantically related data together with the business object logic and the rules required to manipulate the data. Business objects are accessible only via service interfaces.

Note

In contrast to the general notion “object” in an object-oriented sense, the term business object is used here to describe the type and not the instance level. Consequently, the business object SALES ORDER describes the category of all sales orders and not a single instance of a sales order.

2.3.3 Process Component as Reusable Modeling Entity

Before we continue with process components, we have to take a closer look at the following terms: business process, end-to-end business process, and integration scenario:

- A business process is a set of logically related business activities that produce a specific business result. A business process can be restricted to one functional area in an enterprise or can span several of them. A functional area is, for instance, order entry or invoice handling.
- An end-to-end business process is a complete business process from its origin to its end across all functional areas affected, even beyond enterprise boundaries.
- An integration scenario describes the integration aspects between all functional areas touched by an end-to-end business process.

With these definitions in place, we can understand a process component as a reusable building block for modeling integration scenarios (see figure 2-2). The activities performed within one process component equal a business process. An end-to-end business process is a process across several process components.

Process components bundle certain coherent business functionality and comprise one or several business objects. Process components help software architects and developers to decompose overall business functionality into manageable, business-relevant, and meaningful units. The integration example scenario SELL FROM STOCK (see chapter 3.1) contains the process components SALES ORDER PROCESSING and CUSTOMER INVOICE PROCESSING with the business objects SALES ORDER, CUSTOMER INVOICE REQUEST and CUSTOMER INVOICE, among others.

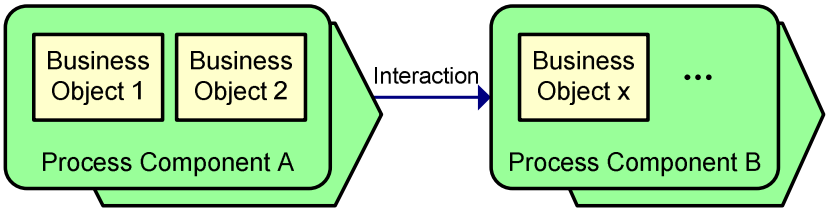


Figure 2-2 A Simple Integration Scenario

2.3.4 Business Process Platform

Enterprise services, business objects, and process components are all embedded in the business process platform (BPP), the fundamental technical and application infrastructure for SAP Business byDesign. In this section we provide a brief introduction to the major architecture building blocks of BPP, namely the application platform (AP) and the technology platform (TP) as depicted in figure 2-3. This figure also reflects the fact that the SAP Business byDesign hat, which includes the SAP Business byDesign user interface together with additional, hat-specific portions of analytics, application adaptation (business configuration), system management, and the built-in learning environment, is a consumer of the services provided by the application platform and the technology platform.

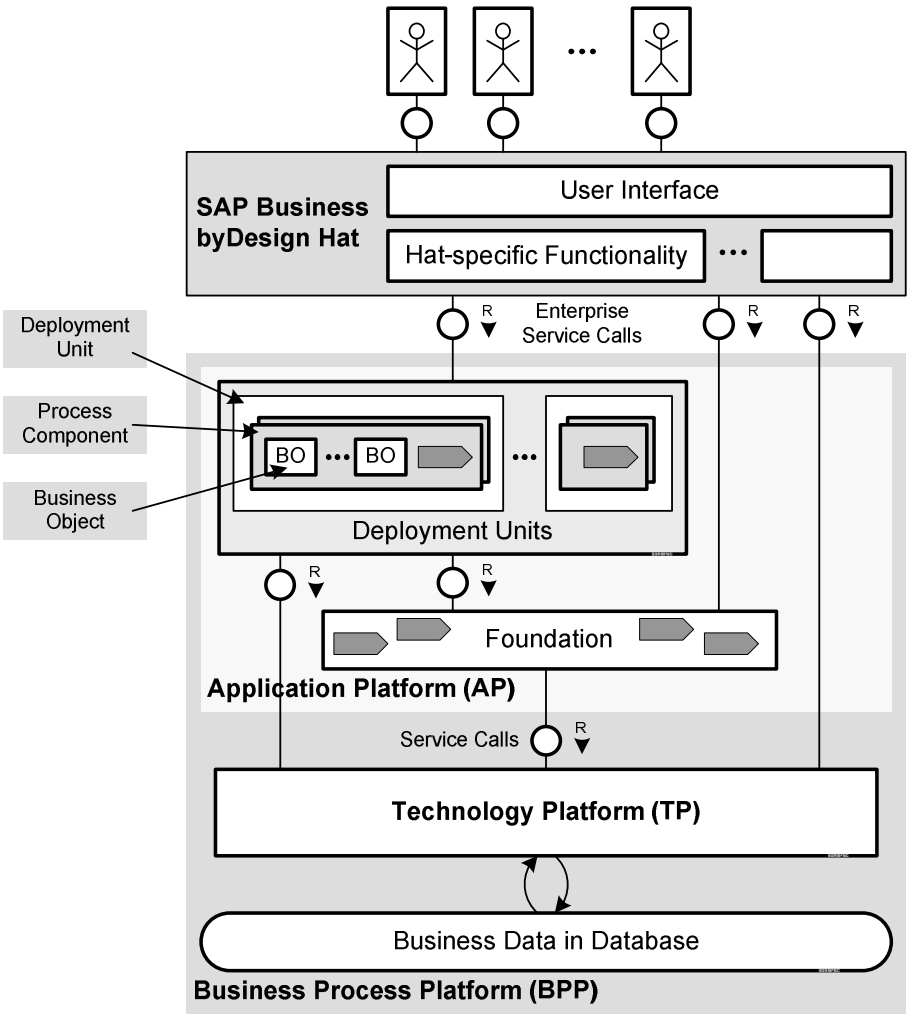


Figure 2-3 Enterprise SOA for Midsize Companies – Runtime Overview

2.3.4.1 Application Platform

The application platform is where core business functionality is implemented. It is structured into one foundation for generic functionality and multiple deployment units for business scenario-specific functionality.

2.3.4.1.1 Deployment Units

A deployment unit groups semantically related process components that need to run together on one system. Multiple deployment units can run together on the same system or on different systems. Thus companies have the following deployment options:

- All deployment units can be operated centrally on one system.

- Deployment units can be distributed on several systems to support decentralized business operation, for example central logistic execution and individual customer relationship management in each sales office.
- Selected deployment units can be outsourced, but they are integrated with centrally operated ones.
- Selected deployment units can be replaced by custom-developed deployment units or other applications.
- AP / ByDesign can be deployed incrementally, meaning that deployment units can be activated step-by-step over time as required.

Communication between different deployment units is handled via enterprise services – typically using asynchronous message exchange. The process agent framework introduced in chapter 5 provides the technical infrastructure for this message exchange.

To simplify the installation process, all deployment units of the application platform are installed at the same time as a whole, and then they are activated subsequently according to an enterprise's specific needs.

2.3.4.1.2 Foundation

Business software solutions are not built solely from business objects and functions specific for a given business scenario; they also include generic entities. These are, for instance, master data, organizational data, or generic reusable functions like date and time calculations. In the application platform, these generic reusable elements are available through the foundation, which is part of every instance of the business process platform.

The foundation can be accessed locally from all deployment units. SAP development as well as development partners apply these generic reusable elements for implementing their specific business processes.

2.3.4.2 Technology Platform

The business process platform includes the entire technical infrastructure required to design, implement, and run service-oriented applications. In the context of this book, it is necessary to understand the following elements of the technology platform:

- Enterprise services infrastructure (ESI), which is the technical backbone for service enablement, service configuration, and monitoring (both design time and runtime). ESI includes the enterprise services repository, which is where process integration, business objects, and service interfaces are modeled.
- Process agent framework for message-based business process integration
- SAP NetWeaver® Portal, Web Dynpro technology, and SAP NetWeaver Visual Composer for user interface design and runtime
- SAP NetWeaver Business Intelligence for analytics and fast search infrastructure for query execution

All elements mentioned are discussed in the subsequent chapters of this book.

2.4 Model-Driven Software Life Cycle

Typically, people with different skills participate in different roles in the construction of an enterprise application, using different concepts and terminology and contributing different aspects to such an application. Experience has shown that no single (programming) language is equally well-suited to all the skill sets of all roles or adequately supports all concepts held by all people involved. This is where modeling come into play.

2.4.1 Models

A model is an abstraction of something that exists or could exist. A model serves one or a few specific purposes; for example, one model of a transistor is required to calculate an electronic circuit, while another serves to organize the mechanical mounting of the transistor on the heat sink. The representation of a model solidly depends on its semantics and on its purpose; typical representation can be text, graphics, formula, or table.

In the domain of software products, models can be used by people in sales, service and support, consulting, and development in order to define, describe, and understand business semantics, integration scenarios, system landscapes and behavior, and technical architectures.

2.4.2 Business Software Life Cycle

From the software vendor's view, the life cycle of business software products has several phases subsequent to requirement analysis and portfolio planning (see figure 2-4):

- Development
  - In the specification phase, the software vendor uses business requirements and portfolio planning to develop a product specification.
  - During design, technical entities such as business objects are specified in detail.
  - In the implementation phase, actual program code and other runtime entities such as user interfaces are created. The result is an executable system.
- Sales
  - The customer analyzes products from various vendors.
  - To choose the product that best fits his needs, he requires a comprehensive description of features, business options, and benefits of the products being considered.
- Implementation and configuration
  - In the implementation project, the product has to be integrated in the customer's system landscape.
  - After the software has been installed, it has to be configured according to the customer's business requirements.
  - The product has to be adapted not only during the initial implementation project, but continuously as requirements change over time.
- Operation
  - Monitoring and regular system analysis can identify problems before they run the system into a critical state.
  - If customers encounter problems, they require quick and direct support from the vendor.
  - The vendor provides product update and upgrade packages to maintain the software and to offer improved features.

Activities during all phases require communication and division of labor among many people, and accessing knowledge from previous activities. For example, design is done by many developers and architects, and the knowledge produced in this phase is vital for configuration activities at the customer's site and for support.

2.4.3 Models Used in the Software Life Cycle

For AP / ByDesign, models play a key role during all phases from development to operation. Models created during development provide useful information for subsequent stages in the software life cycle; they can serve as a basis for more specific models created. For example, a business process model created during development serves as a basis for the configuration of a customer's system and as a reference for the support help desk later on.

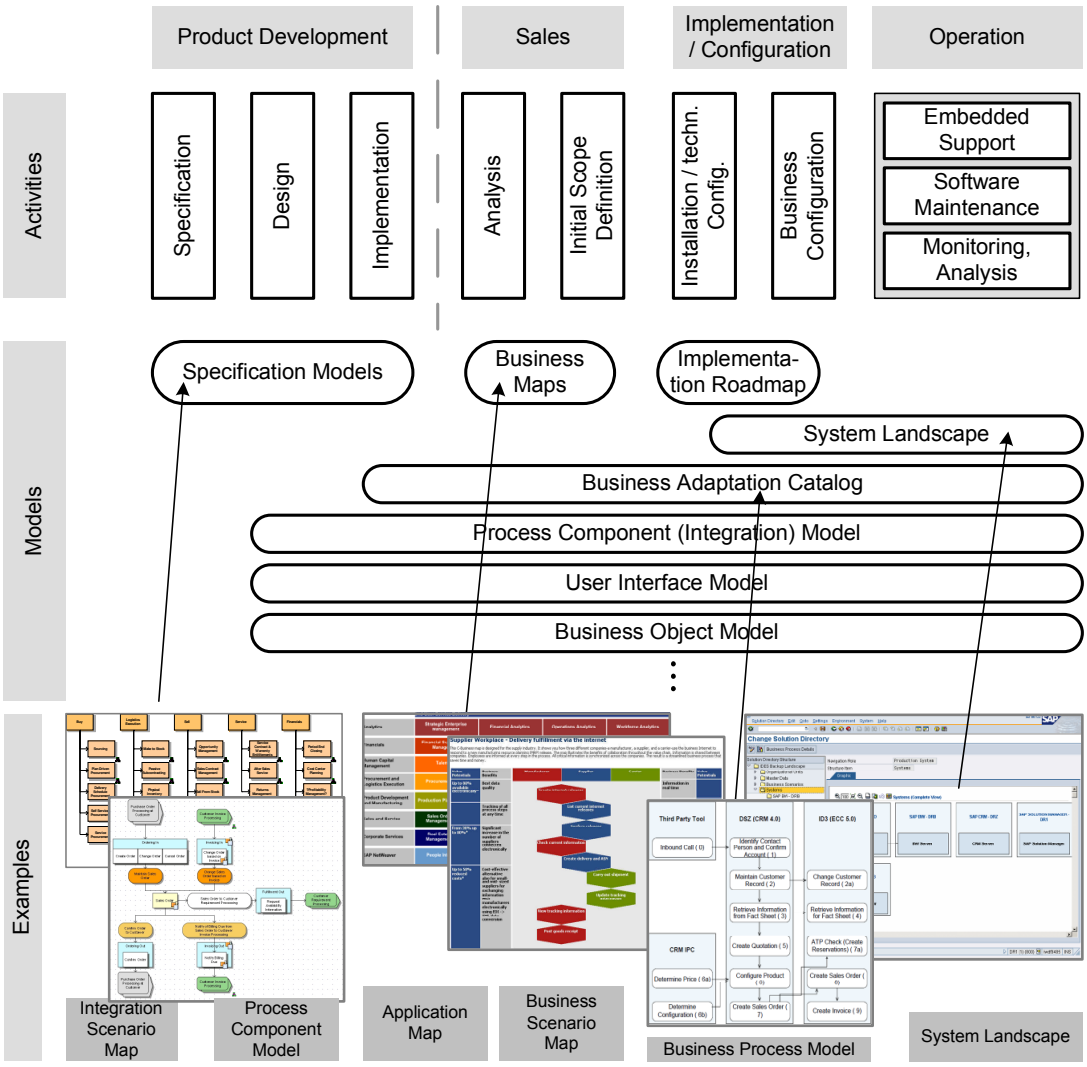


Figure 2-4 Model-Driven Software Life Cycle

A vital prerequisite for a model-driven approach is having access to the models during all phases. Furthermore, they must be consistent with the actual system. Model repositories, such as the ES Repository, and tool support during development, deployment, and operation are necessary for the model-driven software life cycle.

Since the majority of models are created in the development phase, model-driven development is discussed in detail in chapter 10.

2.5 Road Map Through This Book

The purpose of this book is to provide architects, application developers, and user interface (UI) developers with a detailed introduction to the fundamental architectural concepts of SAP Business ByDesign and the underlying application platform. After reading this book, developers and architects will be able to understand the following architecture concepts:

- Business objects and services
- Enterprise services infrastructure and transaction handling
- Message-based integration using process agents
- User interface
- Analytics
- Enhancement techniques
- System operation
- Models used during development

We have structured the book in a way to move quickly from general concepts to more specific topics of interest for architects and developers

After this introduction, we start by explaining the details of the fundamental building blocks of the business process platform: business objects, process components, services, process agents, and message integration.

We then discuss the UI architecture of AP / ByDesign: how the user interface is structured and how it is decoupled from business logic. The concepts of control centers, work centers, and reusable, standardized UI building blocks are explained in detail.

After this we dive into the analytics architecture and show how it helps to provide users with the complete set of information that

they need to monitor business processes and make tactical and strategic decisions.

Then we look at ways to adapt the scope of AP / ByDesign with business configuration, extensions, and composite applications and discuss how to operate the solution.

The book is completed by a chapter decribing the model-driven development process of application platform and SAP Business ByDesign in detail, followed by a summary and an outlook.

Note

Our objective in this book is to introduce and explain main ideas and concepts. To achieve this we concentrate on the logical aspects, preferring simplicity and clarity over full-fledged formal correctness. As a result, the representations you find may not always correspond exactly to what you find in the corresponding tools, the underlying database tables, or the service implementation.

Because enterprise SOA and AP / ByDesign are evolving rapidly, we recommend that you visit SAP Developer Network (www.sdn.sap.com) for up-to-date information; look especially into Enterprise Services Community. For additional resources, refer to the bibliography.

3 Business Objects and Services

In the course of building business software, developers should start with a clear understanding of real-world business processes as basis for defining the semantical entities the business process works on: the business objects. And they should identify the activities performed on the business objects: the services.

In this chapter we explain the details of business objects and how they provide services. To do so, we make use of the integration scenario example SELL FROM STOCK.

3.1 Example: Integration Scenario "Sell from Stock"

The example introduced here is a simplified variant of an integration scenario that is common in many industries: selling goods from stock. This scenario may vary from one enterprise to another and from one industry to another, but the typical sequence of steps is as follows:

1. A company – the customer (buyer) – wants to buy some products from another enterprise – the seller. To do so a purchase order is created in the buyer's system and sent electronically via business-to-business (B2B) communication to the seller.
2. The seller receives the purchase order and creates a corresponding sales order. As part of sales order processing, availability-to-promise (ATP) checks are executed to verify customer requirements; for example whether the requested delivery date can be met and whether there is sufficient stock in inventory.
3. If the seller can fulfill the request, a purchase order confirmation is sent back to the customer.
4. The products are shipped to the customer and the invoice is created.

To use this scenario for explaining key concepts of AP / ByDesign we break it down into the following process components (see figure 3-1):

- PURCHASE ORDER PROCESSING for creating and sending the purchase order (usually this functionality is located in the buyer's system)
- SALES ORDER PROCESSING for receiving the purchase order and processing it as sales order
- CUSTOMER REQUIREMENTS PROCESSING for checking the availability of the requested products in stock and triggering the delivery of the ordered products
- CUSTOMER INVOICE PROCESSING for triggering billing

These process components are depicted in figure 3-1. Each process component represents a reusable building block available for modeling integration scenarios. For example, the process component SALES ORDER PROCESSING can also be used in the integration scenarios CASH SALES or MAKE TO ORDER.

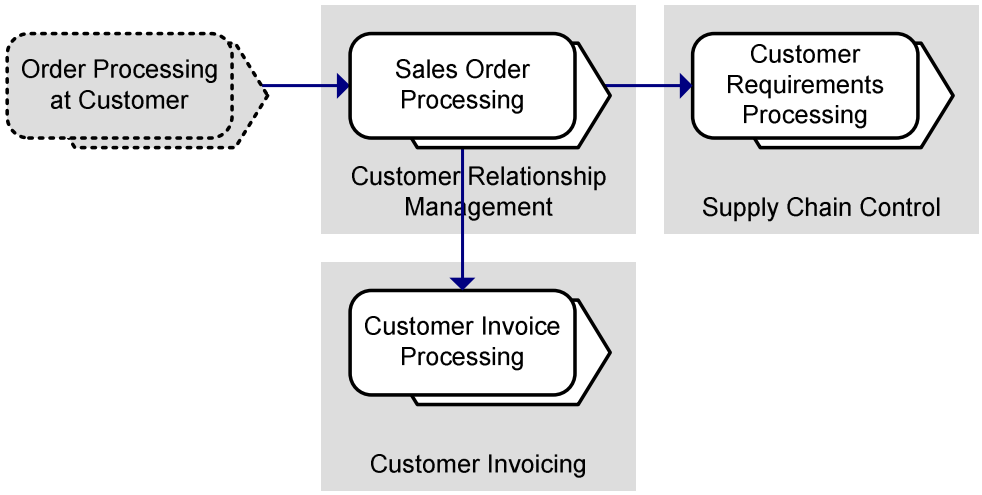


Figure 3-1 Process Components Used in the Integration Scenario Sell from Stock

Note

The integration scenario example SELL FROM STOCK is delivered with every SAP NetWeaver® 2007 system in the software component BASIS, package SESA_SAMPLE. It can be used for demonstrating the concepts of enterprise SOA and for testing.

3.2 Business Objects

A business object, in general, is an entity of significance to a business. In AP / ByDesign business logic and business data are encapsulated in business objects. Business objects act as service providers as well as service consumers and are accessible exclusively through a standardized set of services.

Each business object is described by a business object model, which defines the following characteristics (remember that business object means the type and not the instance):

- Structure of the business object's attributes
- Type of the business object's attributes
- Aspects of the business object instance's behavior during its life cycle
- Service interfaces for accessing the business object's instances

The business data of a business object is described as a set of attributes in enterprise services repository (ES Repository). The attributes are defined by standardized data types, which ensure that attributes with the same semantical meaning always use the same data type. This is crucial since in the application platform all parameters of service interfaces are derived from business object attributes.

Business object models are described in the enterprise services repository solely on meta data level, which means the description is independent from the implementation in a programming language. Only the business object model including the business object's service interfaces is visible for service consumers.

At implementation time, the business object model is used in the selected development environment to create a programming language-specific representation, for example in Java or ABAP™. The corresponding development tools are the SAP NetWeaver® Developer Studio for Java and the ABAP workbench. Because business objects and services of the application platform are implemented in ABAP, we focus solely on that programming language throughout this book.

In ABAP each business object is represented by one class called the service provider class. The business object's program code, which might reside in multiple classes or function modules, can be

reached from outside only via this service provider class. In addition, each business object owns a set of database tables for persisting its data. Business object data can be accessed only via the business object's service interfaces. Direct access to the business object's database tables is not allowed.

In the following sections we take a detailed look on the structure, elements, and behavior as described in the business object model. For details about the implementation of business objects and their services, see chapter 4.

3.2.1 Structure

A business object is structured as a tree of business object nodes with one root node. Each business object node represents a set of semantically related attributes of the business object. The hierarchy of the nodes is defined through associations between the nodes starting with the root node (see section 3.2.1.2.1). For example the business object sales order has a root node for sales order header information and an associated node for sales order item data like the ordered products (see figure 3-2).

3.2.1.1 Business Object Nodes

As explained in the section before business object nodes are used for structuring business object attributes. Each business object node has a unique name within one business object. A data type, the so-called node data type, is assigned to each node, embracing all attributes of the node (see figure 3-2). To enable the identification of a business object node instance at runtime, each node gets a node identifier.

Data of business object nodes is stored in tables of a relational database. To avoid data inconsistency caused by parallel updates of multiple nodes on the database, business objects are modeled without redundancies. Each attribute exists only once within a business object (for exceptions, see chapter 6.3.2.1).

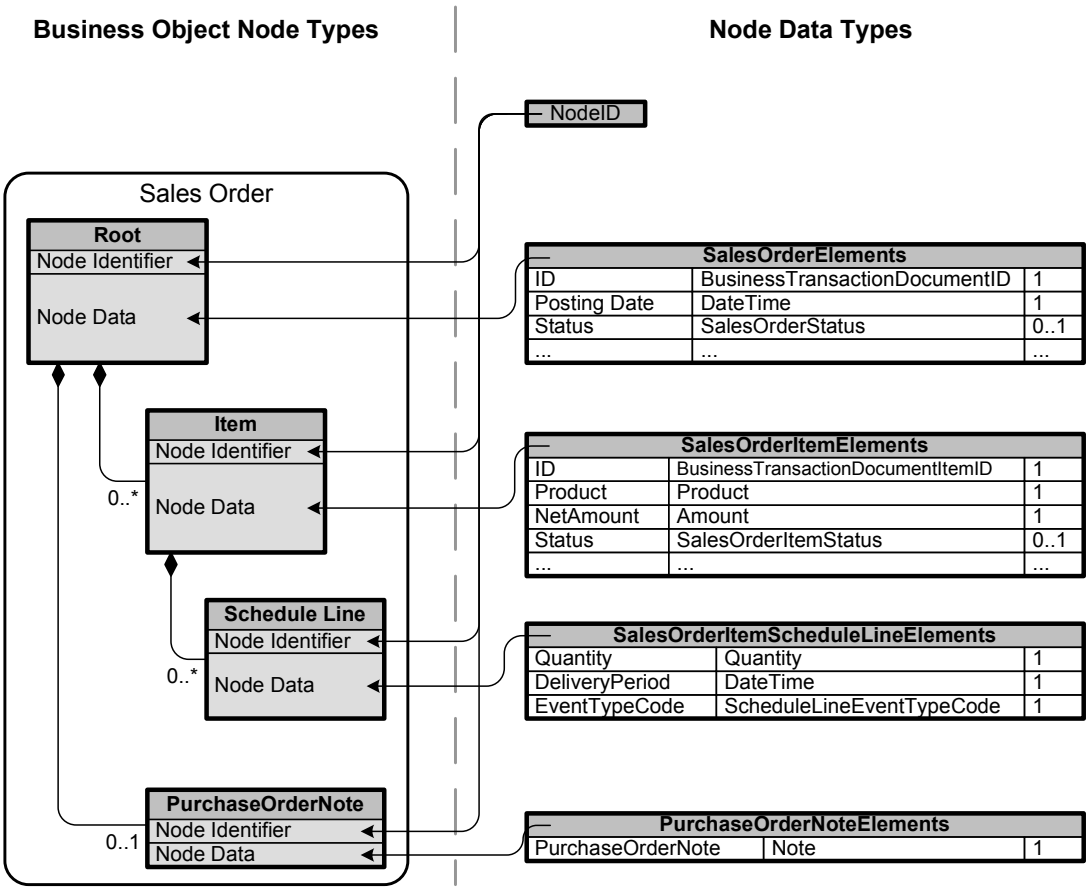


Figure 3-2 Business Object Model with its Tree of Nodes

3.2.1.2 Associations

Business object nodes are not independent but are semantically interrelated, for example by a parent-child relationship. Semantic relationships between business object nodes are defined by unidirectional and binary named associations (see figure 3-3). Associations can be set up between the nodes of one business object or across different business objects.

3.2.1.2.1 Compositional Associations

Associations that link the different nodes of a single business object are called compositional associations because they pull together or “compose” the business object. Compositional association (composition) defines the relationship between parent node and child nodes within one business object and is used to construct the business object’s tree of nodes. A composition represents a strong semantic relationship: one or more instances of the child node depend on the existence of one instance of the parent

node. One instance of a parent node may have zero, one, or multiple associated instances of a corresponding child node.

Example of Compositional Association

In figure 3-3, the business object SALES ORDER contains the root node SALES ORDER, which includes the fields known collectively as sales order header. The root node SALES ORDER has a compositional association to the node SALES ORDER ITEM, thus the SALES ORDER ITEM belongs to the business object. SALES ORDER ITEM itself can have associations to other nodes such as SCHEDULE LINE.

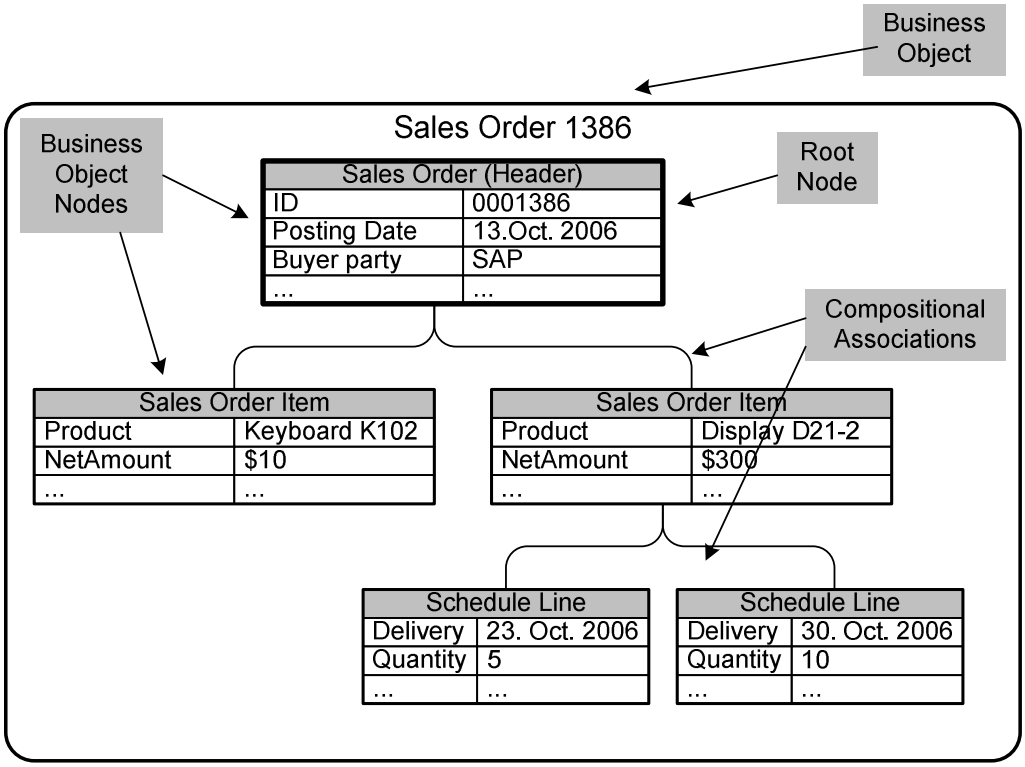


Figure 3-3 Business Object Instance with Compositional Associations

3.2.1.2.2 Cross-Business-Object Associations

If we take a closer look at the business object SALES ORDER of our example, it is obvious that it has to contain information about the ordered product as well as data of the business partner. But BUSINESS PARTNER and PRODUCT are business objects on their own because they are independent business entities. To address this situation, relations between different business objects can be established by using cross-business-object associations.

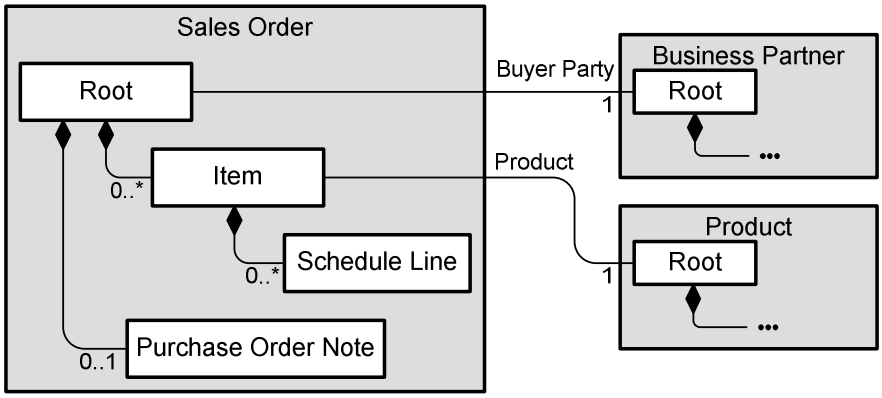


Figure 3-4 Associations Across Business Objects

A cross-business-object association is a relationship between two nodes of two different business objects that reside in the same deployment unit or in a deployment unit and the foundation (see 2.3.4.1). In our example the root node of sales order has a cross-business-object association to the root node of BUSINESS PARTNER for linking customer data to the sales order header. In addition the node SALES ORDER ITEM has a cross-business-object association to the root node of PRODUCT.

3.2.2 Attributes

Each attribute of a business object is defined by a data type. A data type is the formal description of representation, structure, and interpretation of data used by software programs. To homogenize the use of data types, SAP defines consolidated global data types that represent business-related content in conformance with widely used Web and business standards (see section 3.2.2.1).

Especially in the mid-market there is often a high pressure for companies to comply with e-business standards set by bigger companies. Examples are suppliers in the automotive or consumer goods industry. The usage of global data types for business object and service definition opens AP / ByDesign easily up for business-to-business communication. To ensure that global data types are uniform and human-readable, which speeds up the integration at low cost, they are built on the following standards:

- Data type development methodology described in the international standards ISO/TS 15000-5 and UN/CEFACT Core Components Technical Specification (CCTS)

- Extensible Markup Language (XML) schema used for global data type definition

All business objects and service interfaces share the same pool of global data types. This ensures that if the same attribute occurs in different business object nodes or service interfaces, it is always described by the same global data type and the same semantic description. The attributes of one business object node are combined within a node data type. Each attribute within a node data type references a global data type.

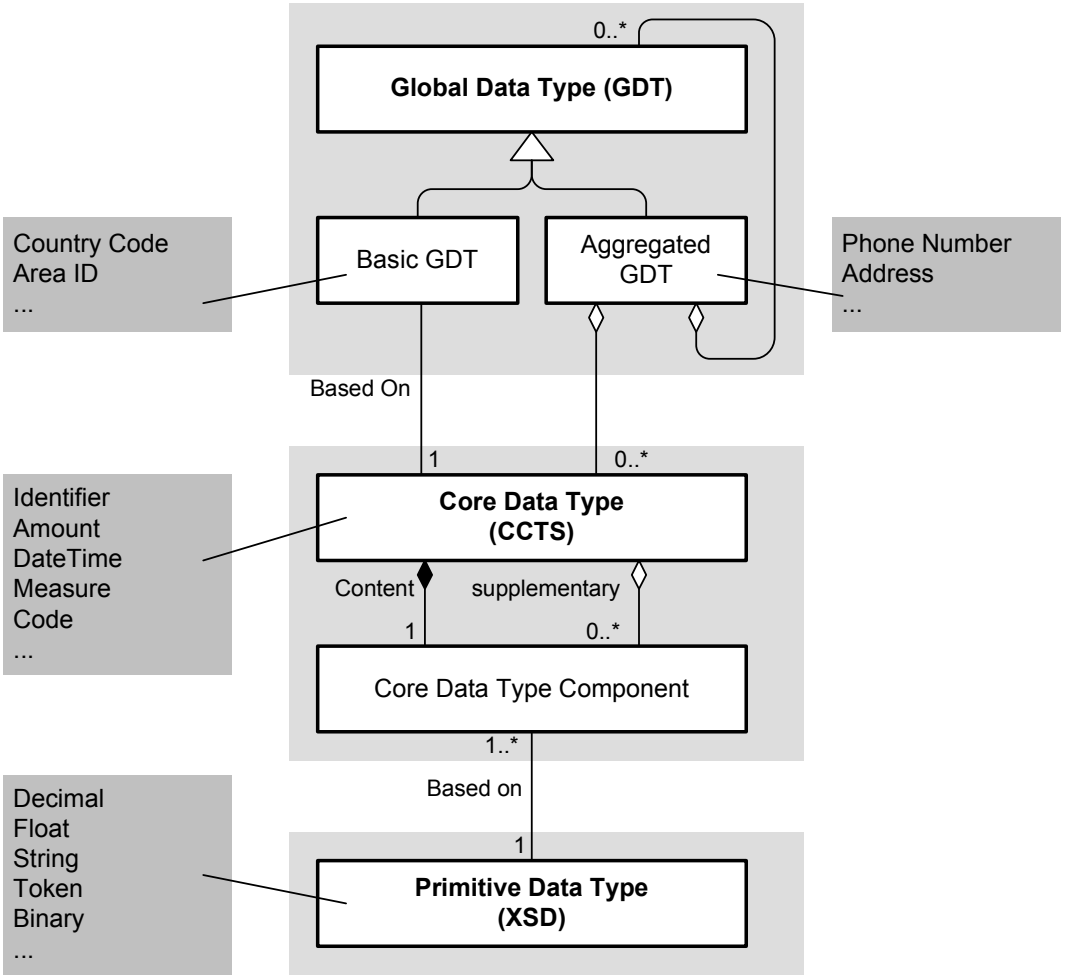


Figure 3-5 Building Blocks of Global Data Types

Global data types are created from core data types (see below), which are standardized data types according to CCTS.

Core data types – in CCTS called core component types – are the building blocks for global data types. They have almost no business semantics but do have a high reuse potential for various global data types. Examples of core data types include AMOUNT,

IDENTIFIER, DATETIME, INDICATOR, MEASURE, NUMERIC, QUANTITY, TEXT, and CODE.

Core data types are typed by primitive data types according to XML Schema Definitions (XSD) defined by World Wide Web Consortium (W3C). These primitive data types are FLOAT, STRING, DECIMAL, TOKEN, BINARY, and so forth.

Core data types are defined by a fixed set of properties that are built-in and cannot be changed. At runtime these properties are used to define the layout of the corresponding fields on the user interface. For example, for the core data type DATETIME a calendar to pick up dates is displayed. The core data type CODE has a drop-down list that displays the codes from which a user can make a selection.

Each core data type consists of the business content or value (for example AMOUNT) and additional information (for example CURRENCY). According to CCTS, the business content part is called content component, while the additional information is called supplementary component.

Example of a Core Data Type

The core data type AMOUNT consists of the content component AMOUNT and the supplementary component CURRENCYCODE. In this case the supplementary component is mandatory as an amount without a specified currency is meaningless. The XML representation looks like this:

```
<Amount CurrencyCode=”EUR”>777.95</Amount>
```

AMOUNT is typed with the primitive data type DECIMAL; CURRENCYCODE is typed with the core data type CODE, which is based on the primitive data type TOKEN. Field length is also specified.

For CURRENCYCODE a code list containing possible values (such as $ for U.S. dollar, € for euro, and so on) is specified.

3.2.2.1 Global Data Types

In contrast to core data types, global data types have business semantics. Each global data type consists of one or more core data types and/or other global data types. This means we have two flavors of global data types: basic global data types, built di-

rectly from core data types; and aggregated global data types, created from other global or core data types.

For example, the aggregated global data type PHONENUMBER (see figure 3-6) comprises the following global data types:

- PhoneNumberAreaID
- PhoneNumberSubscriberID
- PhoneNumberExtensionID
- CountryCode

The first three global data types in the example are built directly from the core data type IDENTIFIER. COUNTRYCODE is a basic global data type based on the core data type CODE.

Example of Global Data Type

The XSD description of values typed with PHONENUMBER looks like this:

```
<PhoneNumber>
         <AreaID>6227</AreaID>
         <SubscriberID>7</SubscriberID>
         <ExtensionID>47474</ExtensionID>
         <CountryCode>DE</CountryCode>
</PhoneNumber>
```

The names of global data types follow the naming rules specified in CCTS, based on standard ISO/IEC 11179-5. Each name consist of up to three parts (terms) that reflect the following aspects of a global data type:

- Object class
  A set of concepts, abstractions, or things in the real world that can be identified within clear boundaries and meanings, and whose characteristics and behavior follow the same rules (examples: phone number, automobile, person, household, and order)
- Property
  A characteristic feature shared by all instances of an object class (examples: country, area, subscriber, color, age, income, and address)
- Representation
  Description of how data is represented; that is, the data type and its value range (examples: an identifier repre-

sented with xsd:identifier or a date represented with xsd:datetime)

In general, the name of a basic global data type is the concatenation of the terms for object class, property, and representation. For example PHONENUMBERAREAID includes the object class PHONENUMBER, the property AREA, and the representation IDENTIFIER. In the specific case where the basic global data type is part of several object classes, the object class term is omitted. For example COUNTRYCODE belongs to the object classes PHONE NUMBER, TAX AUTHORITY PARTY, and PHYSICAL ADDRESS; and consists of the property term COUNTRY and the representation term CODE.

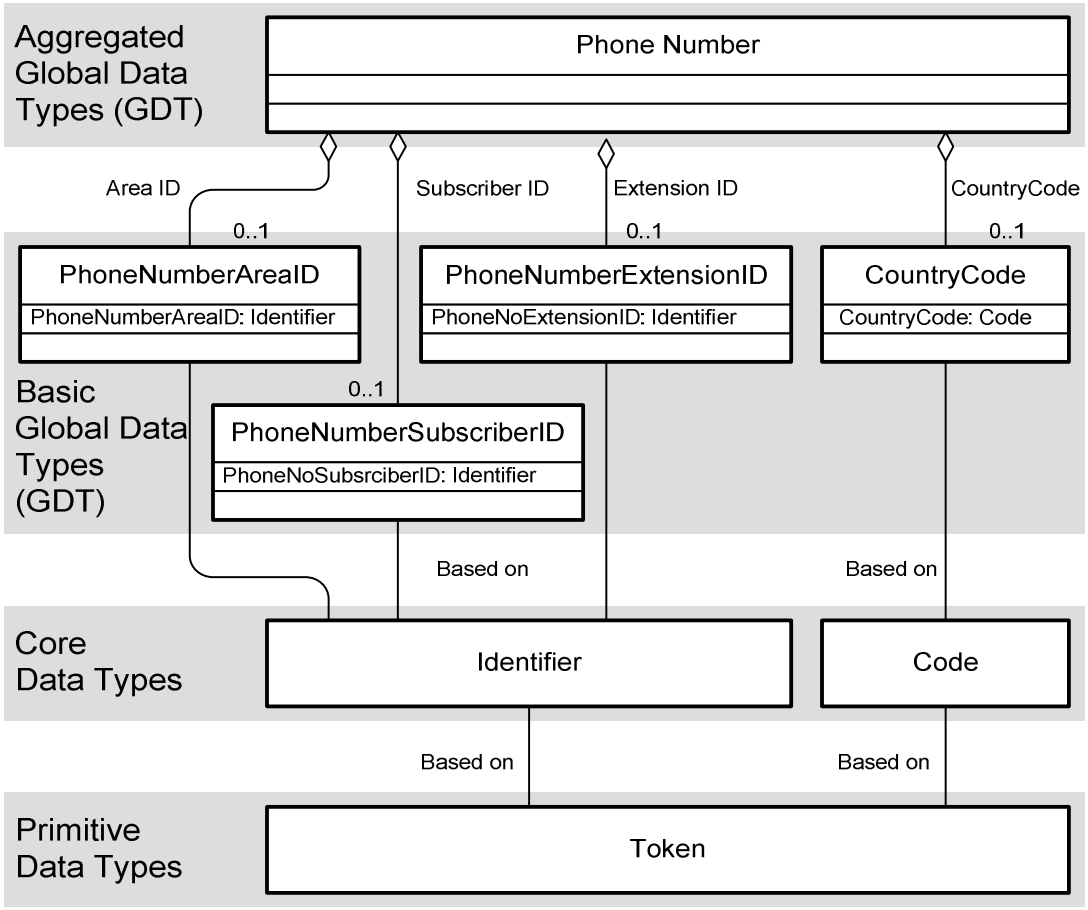


Figure 3-6 Global Data Type "Phone Number"

An aggregated global data type is named by its object class; for example, the global data type PHONENUMBER has its name from the real-world object class phone number. For each element of an aggregated global data type, it has to be defined whether it is mandatory or optional. In addition, the length has to be specified.

All global data types used in SAP software are approved by an SAP governance process for business content. They are stored in the enterprise services repository in the namespace “SAP Global.”

3.2.2.2 Value Sets

A value set defines the allowed values for a given attribute. Value sets are used especially by the user interface to propose input values and to validate data.

As mentioned in section 3.2.2, each single attribute within the node data type of a business object node is described by a global data type. There are two ways to support value sets depending on the underlying core data type.

If a given attribute has the core data type CODE, the possible values are given by a code list. A code list contains the static set of values and descriptions that can be selected for a given attribute. Examples of a code list are a list of currencies, a list of status values, or a list of units of measure.

If static code lists are not sufficient, a business object can define a dynamic value-set service for a data type it uses. The possible values are then calculated by the business object itself at runtime.

Example of Dynamic Value Set

A list of possible actions is defined for working with a SALES ORDER, such as accept, reject, and archive. On the user interface, the drop-down list box for selection displays only the allowed actions for a specific sales order status. The values of this dynamic value set are the allowed actions, which are provided at runtime by a corresponding service of the business object node SALES ORDER HEADER. (For a detailed description of status and actions, see section 3.2.3.)

If the attribute has the core data type IDENTIFIER, the possible values are given by the result of a query service of a business object. This query can be used in the user interface to offer an object search as value help for an attribute type IDENTIFIER.

Example of Object Search as Value Help

The business object node SALES ORDER ITEM has an association to the business object root node PRODUCT defined via the identifier data type PRODUCTID. Because the business object PRODUCT pro-

vides a query service for searching products with the product ID, this service can be used to provide a value help for searching products in the user interface. When a user creates a SALES ORDER, he can look up products to order via this query service.

3.2.3 Behavior

The life cycle of a business object instance covers its creation, various operations on its business data, and finally its deletion or archiving. The behavior of a business object is defined by its business logic, which can be found both in the operations of the service interfaces and in the status and action model.

A status represents a milestone in the life cycle of a business object node instance, reflecting the progress of the business processes that the node instance takes part in. The current status results from the values stored in the status variables of the business object. For example, the value INVOICED of the status variable INVOICING COMPLETION of the root node of SALES ORDER indicates that the invoice has been issued to the customer (see figure 3-7).

Status changes are caused by certain actions (see chapter 4.2.1). These actions represent the possible life-cycle activities performed on a business object node. In our example the action CONFIRM INVOICE changes the status INVOICING COMPLETION from NOT INVOICED to INVOICED. It is triggered when the invoice is sent to the buyer.

The relationships between a business object node's status and actions are defined by a status and action model. This model describes the potential behavior of a business object node. It defines which actions are allowed to be performed if certain status values are reached. In the model constraints are visualized by the connections and arrows between status values and actions (see figure 3-7).

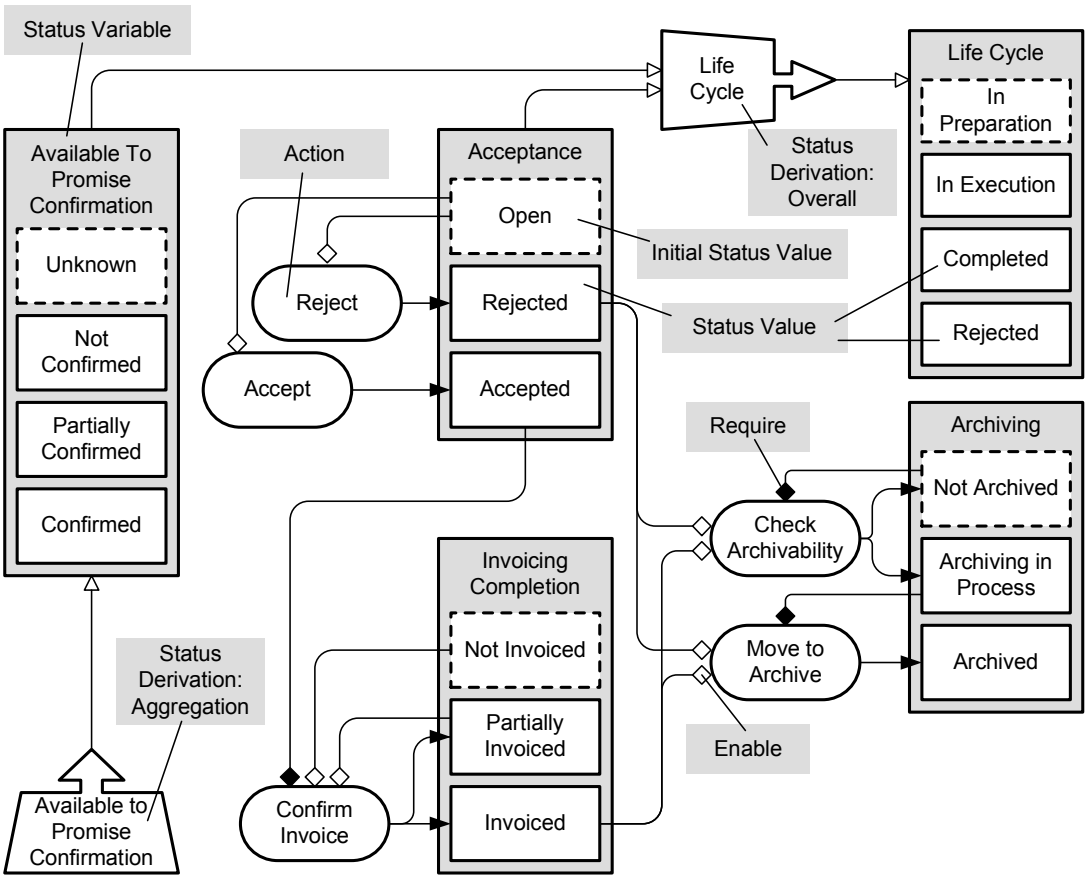


Figure 3-7 Example of Status and Action Model

For each business object node that includes status, a status and action model has to be defined describing the following:

- Status variables and the corresponding list of possible values
- Actions that change status values
- Constraints that define how status values permit or inhibit the execution of an action

Each status variable is defined according to the core data type CODE. Its possible values are specified in the corresponding code list (see section 3.2.2.2). Status variables are part of the node data type of the corresponding business object node.

At runtime the business object checks whether the model allows the execution of a given action for the current status and triggers the appropriate status transition after successful execution. The status and action schema is a passive constraint model in the sense that even if an action is allowed by the model it is not executed automatically.

3.3 Service Interfaces

A business object is accessed only through its service interfaces. The standard for defining service interfaces is the XML-based Web Service Definition Language (WSDL). According to WSDL, each service interface consists of a set of operations (see figure 3-8). An operation is the abstract definition of the smallest discrete callable functionality provided by a certain business object. The input and output parameters of an operation are defined by the types of the input and output messages (see figure 3-8).

A WSDL description of a service contains more information than service interface and operations. Precise details on how and where the functionality is offered can be found in the binding section of the WSDL, with descriptions of protocols, transmission formats, and endpoint information. A consumer must have this information to call a service at runtime.

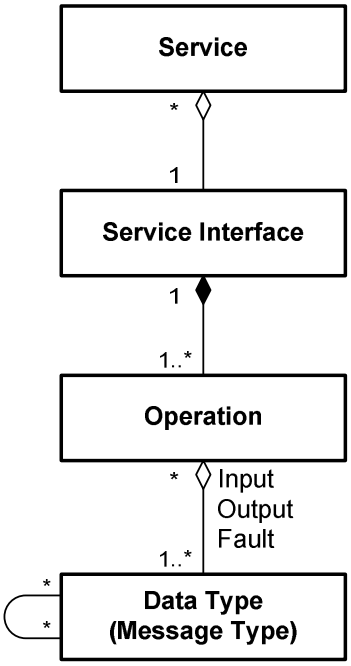


Figure 3-8 Service Definition according to WSDL

3.3.1 Enterprise SOA Metamodel

With enterprise service-oriented architecture (enterprise SOA), SAP added the business aspect to WSDL by creating the enterprise SOA metamodel as depicted in figure 3-9. This metamodel is made up of the following parts:

- The service description in the middle of the figure represents services as defined by WSDL. The underlying message types are referencing global data types.
- On the left side of the figure, the business object defines the meta structure of business objects, including nodes and node data types. Business objects are assigned to process components.
- Shown on the right side, interface and communication patterns guide the definition of service interfaces and operations (for interface patterns see chapter 3.3.3; for communication patterns see chapter 5.3.4.2).

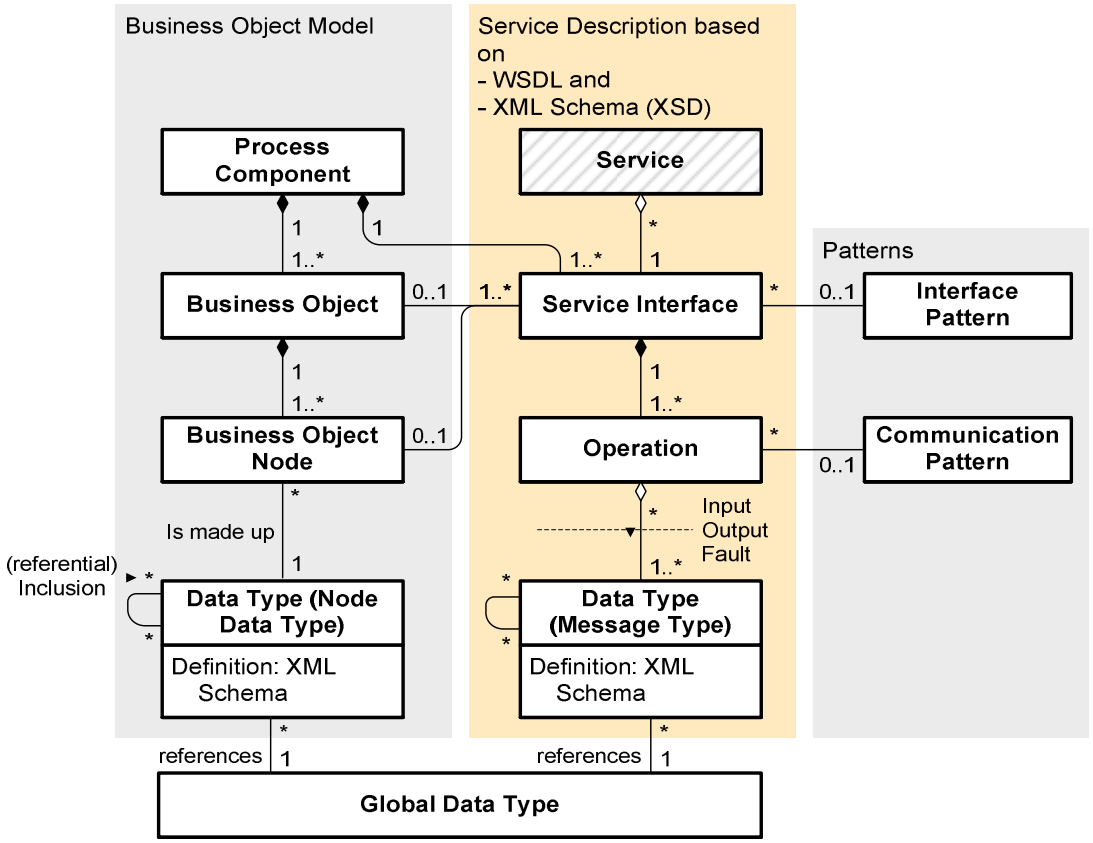


Figure 3-9 Enterprise SOA Metamodel

Business objects and services adhere to the same common metamodel and share the same global data types. The metamodel is the basis for specification and design of all services and business objects.

3.3.2 Local and External Service Consumers

Services are accessed not only locally, for instance by the user interface or other local business objects, but also from external service consumers like other applications or other deployment units. Local and external service consumers require different service granularity. To accommodate this, two types of services have been introduced (see figure 3-10)[1]:

- Core service for accessing a business object node
- Compound service for accessing one or more business objects

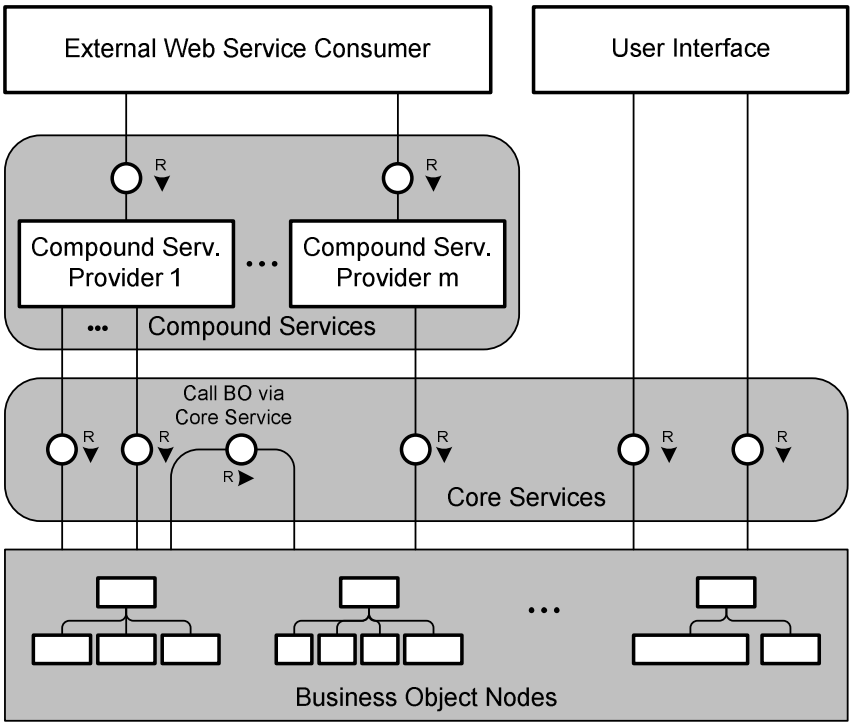


Figure 3-10 The Interplay of Compound and Core Services

Compound services do not access business objects directly; instead, they use core services to do this. This leads to a layered approach. The business objects provide fine-granular core services with generic operations on a node level. Compound services provide specific operations and parameters accessing one or more

[1] Services, whether core or compound, that are published by SAP for external usage are called enterprise services (see chapter 2.3.1).

business objects. They are used especially for communication between different deployment units.

In the following section we cover core services; compound services are covered in section 3.3.4.

3.3.3 Core Service Interfaces

Core services provide a generic interface to the business logic of business object nodes. Like SQL has standardized access to database tables, core services standardize access to business objects, or to be more precise, business object nodes. A set of service operations with very basic functionality, such as READ DATA, MODIFY DATA, and SAVE DATA, was identified through analysis of existing business applications. In order to standardize access to business objects, these service operations are provided by each business object node in a unified fashion.

To accomplish this, interface patterns are defined. They combine semantically related operations that are implemented by each business object node in the same way. For example, one of these interface patterns is the ACCESS pattern, which clusters the operations CREATE, RETRIEVE, UPDATE, and DELETE (see figure 3-11). Each business object node provides exactly these operations for accessing and modifying its instances.

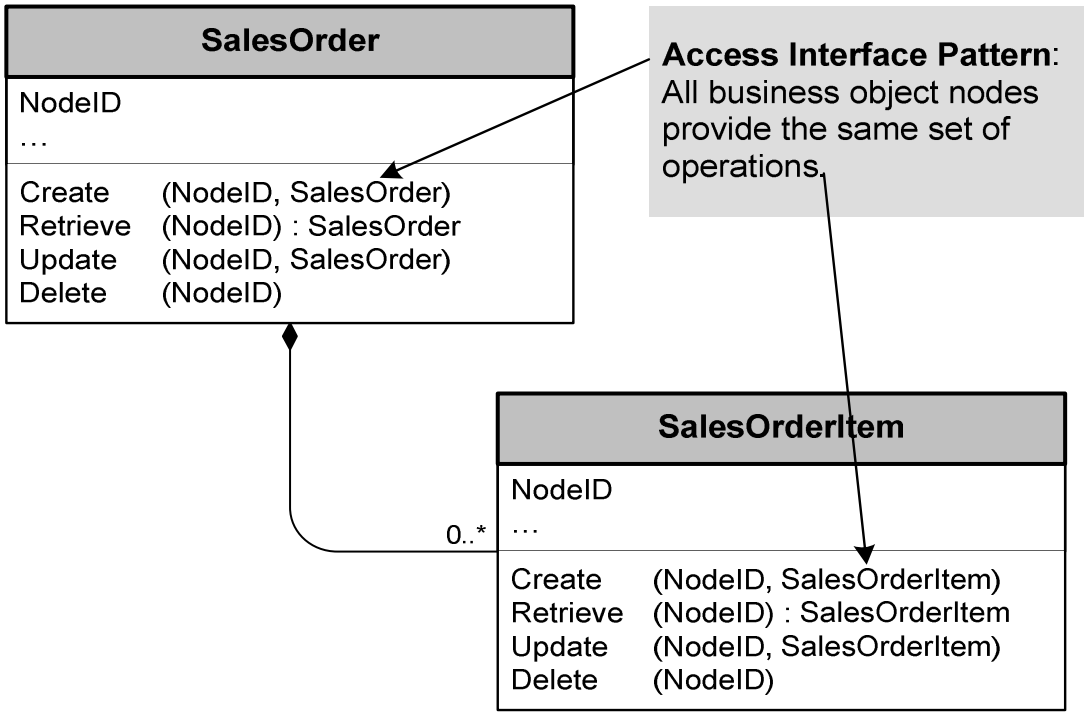


Figure 3-11 The Access Interface Pattern

The following interface patterns for core services have been defined:

- ACCESS interface pattern for reading and manipulating the data of a business object node instance, including operations for create, retrieve, update, and delete
- QUERY interface pattern providing the query operation for searching business object node instances
- ACTION interface pattern provides the action operation for triggering actions. Actions are used to change status or trigger specific activities, like performing checks or printing.
- TRANSACTION interface pattern for controlling the lifetime and transactional behavior of business objects nodes, including the SAVE operation for database updates
- VALUESET interface pattern for retrieving dynamic value help for either the selection of business object node instances or attributes (see section 3.2.2.2)

The advantage of this pattern approach is that all business objects provide the same interfaces and can be accessed in the same way. This makes it possible to build, for example, configurable user interface building blocks (see chapter 6.1.2) that can access all types of business objects independent of specific application contexts.

In the following section we take a closer look at each interface pattern.

3.3.3.1 Access Interface Pattern

The access interface pattern contains the following core service operations to perform basic access and manipulation operations on business object node instances.

- Create
  The CREATE core service operation creates new business object node instances. For each new node instance a business object node ID is created.
- Retrieve
  The RETRIEVE core service operation retrieves the business object node data for a given set of business object node IDs. Depending on the input parameters, it reads the data either from the database or from main memory. The RETRIEVE operation includes a parameter for setting differ-

ent edit modes (read-only, edit, or edit exclusive), which allows the service consumer to signal to the provider whether and how the retrieved data will be edited. This allows the service provider to implement locking policies for concurrency control if requested.

- Update
  The UPDATE core service operation changes the data of existing node instances in main memory. The update is persisted to the database during the SAVE operation of the transaction interface pattern.
- Delete
  The DELETE core service operation deletes existing node instances and their dependants in main memory. The update is executed by the database during the SAVE operation of the transaction interface pattern.
- RetrieveByAssociation
  The RETRIEVEBYASSOCIATION core service operation navigates from source node IDs to target node IDs along a given association.
- RetrieveRootNodeID
  For each non-root node instance, the RETRIEVEROOTNODEID core service operation returns the ID of the root node instance.
- ConvertKeyToNodeID
  This operation is used to find the node instances that correspond to a given key, and return their node IDs.
- RetrieveProperties
  The RETRIEVEPROPERTIES core service operation returns dynamic properties for the requested node instances (see chapter 4.3.3). For example, if modification of a node instance is prohibited, RETRIEVEPROPERTIES returns READ ONLY.
- Check
  The CHECK core service operation checks for the consistency of business object node data before it is written to the database.

Because each business object node provides the operations of the access interface pattern, data access and manipulation happen in a standardized way.

3.3.3.2 Query Interface Pattern

The query interface pattern contains a single core service operation, namely QUERY, which returns a list of node IDs based on given selection criteria and sorting options. QUERY operates on persistent business object data in the database. The core service operation RETRIEVE is used to read the data of the found node instances. For speeding up query execution, external search engines can be integrated using the fast-search infrastructure (see chapter 7.2.1).

3.3.3.3 Action Interface Pattern

The action interface pattern contains a single operation, the ACTION core service, which is a generic operation for executing all actions of a business object node. Actions are defined for each node within the business object model (see section 3.2.3). If an action is about to change a node instance's status, the status and action model determines whether the action can be executed. If applicable, the ACTION core service performs the corresponding status transitions.

3.3.3.4 Transaction Interface Pattern

The core services in the transaction interface pattern are used to control the lifetime and transactional behavior of business object instances. It contains the following core service operations:

- Save
  The SAVE operation triggers the saving of all relevant business object instances to the database and commits the transaction.
- Cleanup
  The CLEANUP operation is called by the enterprise services infrastructure to trigger that business object instances are cleared from the buffers, the corresponding locks are removed, and external resources are released.

For details on transaction handling see chapter 4.5.

3.3.3.5 ValueSet Interface Pattern

The ValueSet interface pattern is used to retrieve dynamic value sets. For an element based on the core data type CODE, this core

service retrieves the possible code values. For an element based on the core data type IDENTIFIER, the service retrieves the possible node identifiers according to the given input (see chapter 3.2.2.2).

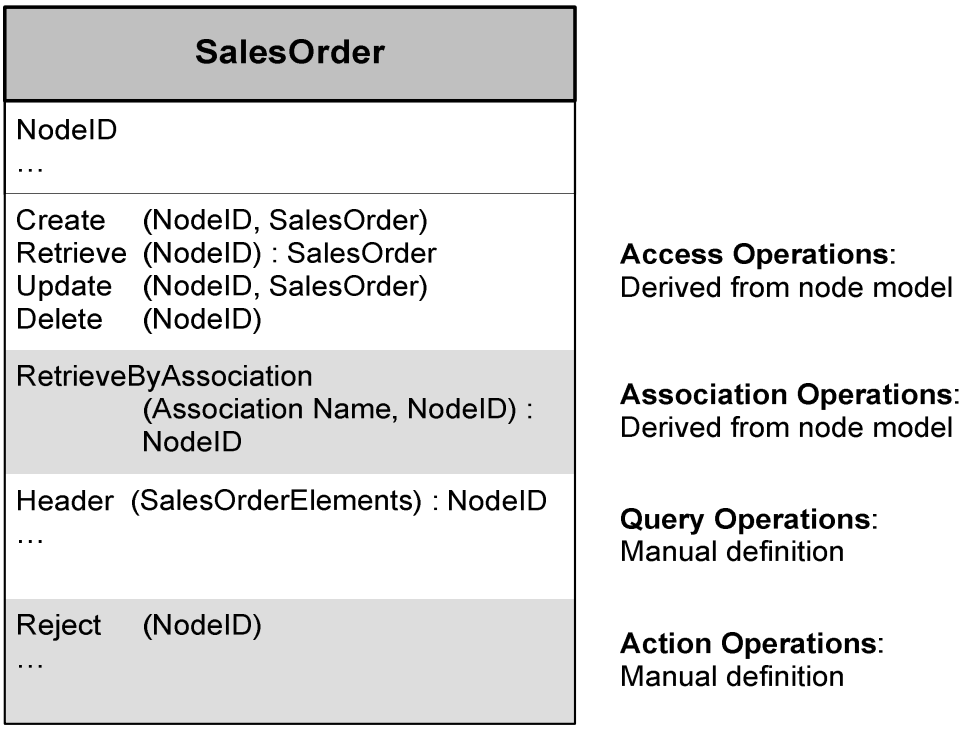


Figure 3-12 Logical View of Important Core Service Operations of a Business Object Node

3.3.4 Compound Service Interfaces

The message exchange between process components in different deployment units within one enterprise is referred to as application-to-application communication (A2A), whereas the data exchange between process components and external application programs across enterprise boundaries is called business-to-business communication (B2B, see also figure 3-1). Typically B2B communication is based on semantical communication protocols like RosettaNet or EDIFACT, which are defined by international standards organizations.

A2A and B2B communication requires coarse-grained services, which are different than fine-granular core services. Therefore we define dedicated compound service interfaces, which typically provide access to a selected subset of the attributes of one or several business object nodes. In this case these attributes can even belong to the nodes of different business objects.

Compound services access business objects only via core services. Therefore we can think of these services as being an aggregation or a “compound” of core services; hence the name compound services (see figure 3-10).

An example of a compound service interface is the interface SALESORDERPROCESSINGORDERINGIN containing the B2B operations for the business object SALES ORDER. These operations include creating, changing, and canceling a complete SALES ORDER, rather than only singular nodes. For accessing the sales order’s business object nodes, the compound service uses the standard core services of the sales order’s business object nodes: CREATE, RETRIEVE, and UPDATE.

3.3.4.1 Inbound and Outbound Interfaces

Compound service interfaces are used for message exchange between service consumers and service providers. While core services interfaces are solely defined by the service provider, compound service interfaces are supplied either by a service consumer or a provider, as follows (see figure 3-13):

- The service provider supplies inbound interfaces containing the operations for incoming messages.
- The service consumer supplies outbound interfaces that define outgoing messages.

The explicit definition of inbound and outbound interfaces makes the service consumer independent of the service provider.

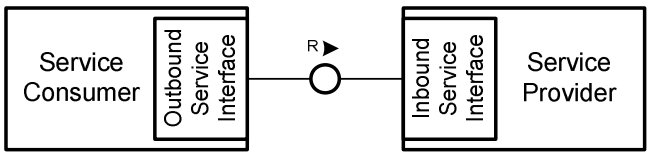


Figure 3-13 Inbound and Outbound Interface of Compound Services

Within the outbound interface the service consumer explicitly defines the structure and elements of the messages it can send. The interface indicates the service consumer’s expectation about the service provider’s interface. On the other side, the service provider specifies within the inbound interface which messages it is able to process. Due to the explicit description of outgoing and incoming

messages it is possible to choose a different service provider for one service consumer and vice versa, as long as the interfaces can be mapped onto each other. This independent definition of inbound and outbound interfaces is the foundation for flexible combination of service consumers and providers. This approach allows, for example, deployment units of different releases of AP / ByDesign to communicate.

3.3.4.2 Parameters of Compound Service Operations

Compound service messages are divided into three categories:

- Request messages include input parameters of an operation and are sent from consumer to provider.
- Response messages include output parameters of an operation and are sent from provider to consumer.
- Fault messages contain error information of an operation and are sent from provider to consumer.

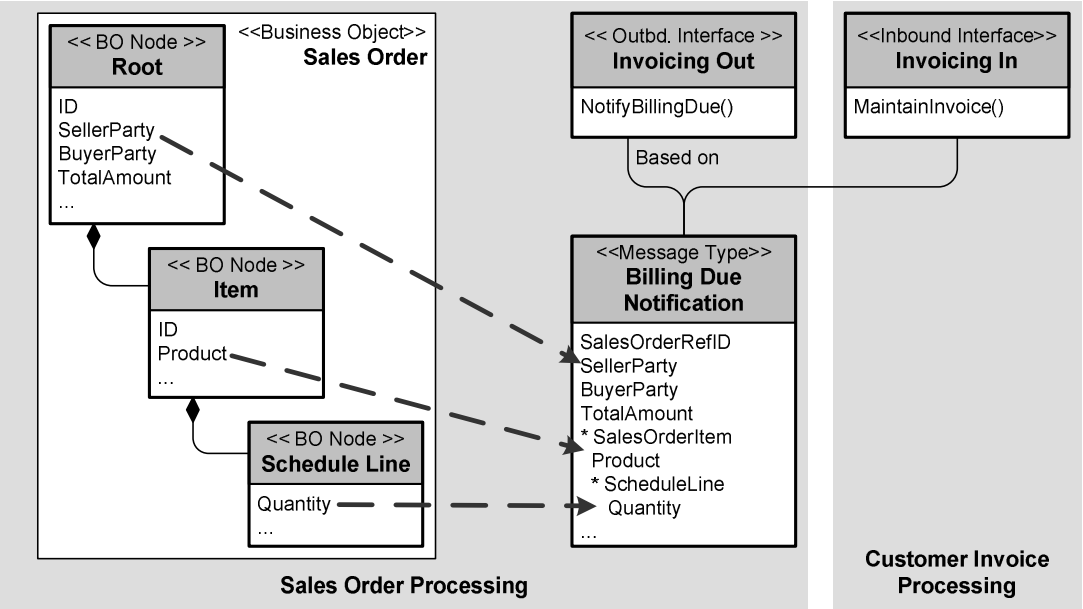


Figure 3-14 Creation of Compound Service Interface from Business Object Model

Each of the three categories is defined by a structured data type, the message data type. Message data types are defined in the enterprise services repository. Within AP / ByDesign message data types for corresponding outbound and inbound operations are usually the same to ensure equally typed parameters. The message data type is composed from individual attributes that are

taken from multiple business object nodes and thus are typed by global data types (see figure 3-14).

For example, the outbound operation NOTIFYBILLINGDUE and the inbound operation MAINTAININVOICE are defined by the same message data type BILLINGDUENOTIFICATION.

For detailed information about the implementation and execution of compound services, see chapter 4.4.

3.4 In a Nutshell

Business objects encapsulate all business logic and business data. Their structure, attributes, and behavior are described within a business object model defined in the enterprise services repository. A business object consists of a tree of associated business object nodes. Each node embraces a set of attributes. The behavior of a business object is specified by its business logic and its status and action model.

Business objects follow a strict programming model. They are implemented by a service provider class and are accessible exclusively through a standardized set of core services. Dedicated interface patterns – access, query, action, transaction, and value set – define the core service operations to be provided by each business object.

A2A and B2B communication requires another type of services. Compound services, which typically provide access to a selected subset of the attributes of one or several business object nodes, open up deployment units for external communication. Compound services use core services for accessing business objects.

Attributes of business objects and compound service parameters are defined using global data types, which represent business-related content in conformance with widely used Web and business standards.

The relationships of business objects, services, and global data types are described in the enterprise SOA metamodel.

4 Understanding Enterprise Services Infrastructure

The key service-enabling technology component of business process platform is the enterprise services infrastructure (ESI). ESI provides design-time tools and runtime engines for creating, providing, and invoking services. ESI enables the interaction of service providers and consumers in a way that they share service definitions without needing to know the details of the service implementation.

4.1 Enterprise Services Infrastructure

ESI plays different roles during design, implementation, and runtime. We explain these roles in the following sections.

4.1.1 Design Time

At design time, the key element of ESI is the enterprise services repository (ES Repository). Business objects, service interfaces, and data types are defined in ES Repository according to the enterprise enterprise SOA metamodel (see 3.3.1) in a way that is independent of the programming language.[2] Thereby core service interfaces are modeled as part of the business object, whereas compound service interfaces are defined as separate entities.

In ES Repository, global data types are used for standardizing semantical information. For business object nodes and compound service operations, structured data types created out of global data types are used. In the case of business object nodes, these structured data types contain all related attributes and are called node data types (see chapter 3.2.1.1). For compound service operations, these data types are called message data types, combining all elements of the message (see chapter 3.3).

At design time developers perform the following tasks in ES Repository:

[2] ES Repository has evolved from the integration builder and integration repository of SAP NetWeaver® Exchange Infrastructure.

- Define data types that can be used for business object nodes or compound service operations
- Define business object models including business object nodes, associations, core service operations, and status and action models
- Define compound service interfaces and operations

After these steps, business object models, service definitions, and data types are available in ES Repository in an implementation-agnostic fashion.

4.1.2 Implementation Time

To implement business objects and compound service interfaces, definitions stored in ES Repository have to be transformed into representations of the target programming language. For this purpose, ESI generates the code skeleton of the service provider class for each business object. For each compound service interface, a proxy class is created, enabling the communication via this interface.

To assist developers in generating these proxy classes, an ES Repository browser and a generator have been integrated into ABAP™ development workbench and the SAP NetWeaver® Developer Studio tool. Developers browse ES Repository content and create code skeletons for selected entities. After generation, the developer supplements the code skeletons of service provider classes with additional program code as needed. At runtime, the proxy classes and service provider classes are executed in the backend without further access to ES Repository.

Example of a Service Provider Class in ABAP Development Workbench

When a service provider class for a business object is generated in the ABAP Development Workbench, the following entities are automatically created from the ES Repository definition:

- ABAP class with methods that are used to implement the core services
- ABAP data types for the ES Repository data types
- Business object metadata including information about nodes and associations

4.1.3 Runtime

At runtime, ESI provides all functions necessary for calling and executing services, exchanging messages, and managing transactions. For each business object, ESI creates an instance of the corresponding service provider class, which thenceforward acts as service provider for all core services of all instances of this business object. For each compound service, the corresponding proxy class is instantiated as well.

For calling a service, ESI supports the following communication modes:

- Synchronous service calls, which block the service consumer until the service call returns
- Asynchronous service calls, which do not block the service consumer while waiting for the service provider to provide a response. If a response has to be sent, a separate service call has to be issued from the service provider to the service consumer.

A service can be stateless or stateful. A stateless service maintains no state information between service calls, whereas stateful services keep state information in main memory across multiple invocations.

In the application platform, compound services can be designed for synchronous or asynchronous communication. They are stateless since they normally fulfill a complete, self-contained piece of work. No state needs to be kept in main memory between multiple invocations. Core services, on the other hand, are fine-granular services. For instance, in the case of an end user who is completing a business task or activity via the user interface, a coherent series of core services needs to be called. For that reason core services are stateful. They are always invoked synchronously in order to present the result of the operation immediately to the consumer.

4.2 Implementation of Core Services

The concept of core services as explained in chapter 3 represents the design-time view. From the logical perspective, each business object node provides a standardized set of core service operations (see figure 4-1). However, this is not the best way to implement

core services. Usually the nodes of a given business object are strongly dependent from a business point of view. For example, attributes of one node can be modified only if attributes of another node have specific values. Or several nodes need to be updated in parallel to reach a consistent state of the business object instance.

To address this, each business object as a whole is represented by one service provider class. All core service calls to this business object's nodes are routed through this class. All core service operations of the same category (query, retrieve, update, action, and so on) of all nodes of one business object are implemented in one method of the service provider class. Thus the service provider class contains one query method for performing all queries, one retrieve method for performing all retrieve operations, and so on.

To ensure that each service provider class implements the same set of core service operations, interface patterns have been introduced (see chapter 3.3.3). Interface patterns are implemented by service provider classes. For example, the core services to CREATE, RETRIEVE, UPDATE, and DELETE are combined in the ACCESS interface of the service provider class (see figure 4-1).

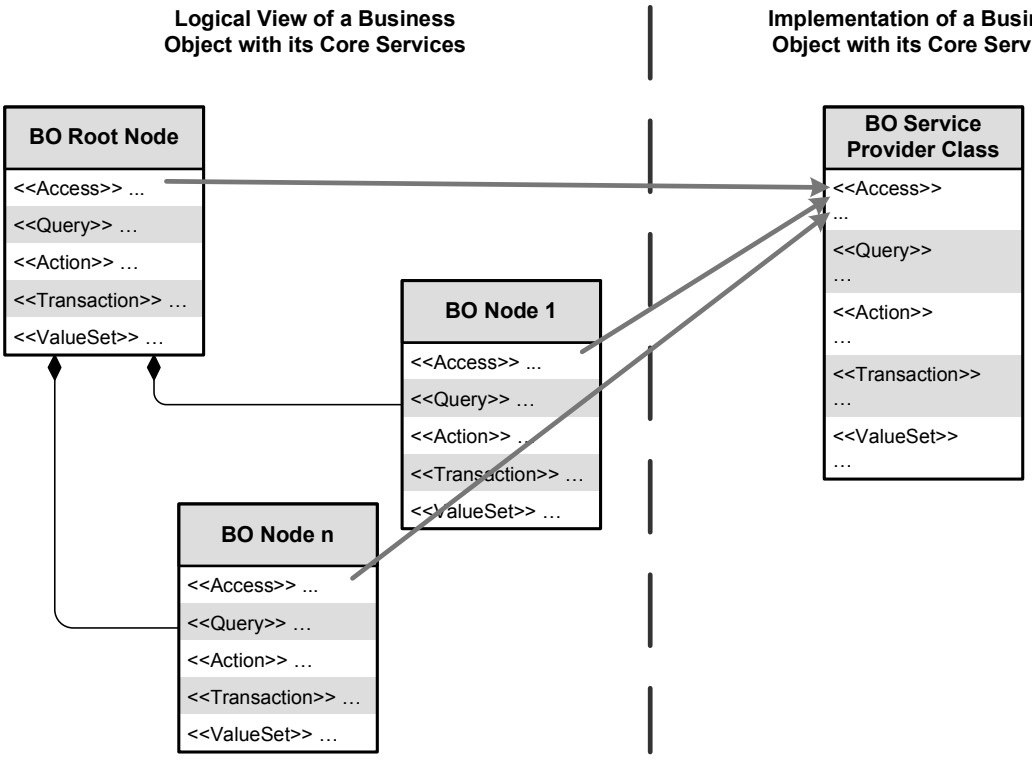


Figure 4-1 Core Service Implementation

Core services provide the advantage of standardized access to business objects (see chapter 3.3.3). This means that a core service provides a generic signature that is not specific to a business object or to a business object node. Each core service operation is called in the same way using the following parameters:

- Node name for identifying the target of the service call
- Node ID specifying the instance of the node
- Input or output parameter

Example: To display the sales order header of a sales order on the user interface, the core service call looks like the following:

```
retrieve(input: node name = “sales order header”;
         node ID = 2343;
         output:result = sales_order_header_data)
```

4.2.1 Action and Query

The core service operations of both the access and the transaction interface pattern (such as CREATE, RETRIEVE, UPDATE, DELETE, and SAVE) describe standard functionality provided by every business object node. In contrast, actions are activities that are specific for a given business object node. Examples are checks that lead to status changes, triggers for follow-up activities, and functions such as printing or archiving (see also chapter 3.2.3).

As explained above in section 4.2, each core service operation is implemented by one method for all business object nodes. This is also true for ACTION and QUERY, which, however, can be shaped individually per business object node at design time in ES Repository.

The implementation of an ACTION core service operation is a generic operation that triggers all actions of a business object node. In addition to the business object node name and ID, it takes as input the name of the specific action that should be executed and action-specific parameters.

For actions that change a status, the corresponding method of the service provider class has to be implemented according to specific programming rules, which take the business object’s status and action model into account (see chapter 3.2.3). At runtime the ACTION method has to call status and action management which interprets the status and action model to determine whether the

action can be executed. At the end of a successful execution of an action, status and action management is called again to perform the necessary status transitions.

The situation is similar for the QUERY core service. In ES Repository several queries with different search parameters can be defined for each business object node. The QUERY core service operation is a generic operation that executes all queries defined for a business object node. Each call of a specific query against a business object node results in a call of the generic QUERY operation with the name of the specific query as input parameter. The QUERY returns a list of business object node IDs according to the given search criteria. The node IDs are used as input parameters for the retrieve operation. The database selections for the defined queries can be implemented and executed within the query method or the service provider class or by using fast search infrastructure (see chapter 7.2.1). In this case, an external search engine speeds up query execution.

4.2.2 Updating Business Objects

The three core service operations for the manipulation of business object node instances – CREATE, UPDATE, and DELETE – are implemented in one MODIFY method to enable collecting of updates. This creates several advantages.

ESI collects CREATE, UPDATE, and DELETE calls of one user interaction and flushes them to the service provider simultaneously. Consequently, the service provider does not have to deal with individual CREATE, UPDATE, and DELETE calls, but can work with batches of such changes.

Concurrent updates of multiple nodes of one business object are often dependent on each other. From a business perspective these updates must be collectively consistent. The single MODIFY method allows the service provider to perform consistency checks before applying the changes on multiple nodes in parallel. In this way the service provider can resolve dependencies between different updates. For example, two separate updates that would result in an error if executed sequentially can be applied in one step.

4.3 Execution of Core Services

Core services are stateful and called synchronously. Depending on the use case, we differentiate between local core service calls and core services calls from the user interface. For each option, ESI provides a specific type of consumer proxy: the local consumer proxy (LCP) for local calls and the remote consumer proxy (RCP) for calls from the user interface. When calls are received via a proxy, ESI's service manager maps the calls to the methods of the corresponding service provider classes.

4.3.1 Local Core Service Calls

Two local core service call scenarios are handled by LCP (see figure 4-2):

- A business object communicates with another business object within one deployment unit
- A compound service accesses a business object instance via core services

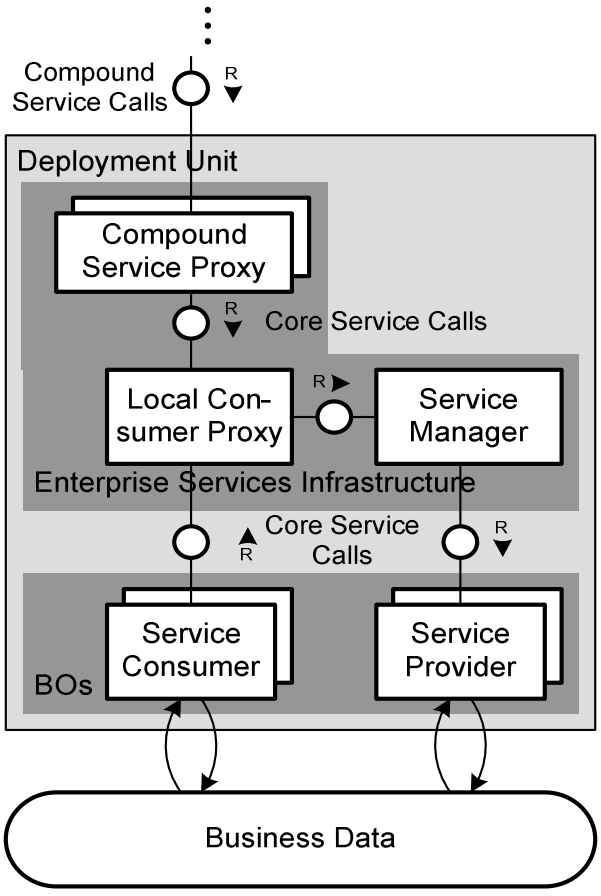


Figure 4-2 Calling Core Services via Local Consumer Proxy

LCP abstracts from the concrete methods of service provider classes and offers the generic core services interface as described in chapter 3.3.3. To call a core service, the service consumer needs to know only the business object node ID. If a service consumer calls a core service, LCP forwards this call to the service manager, which invokes the method of the corresponding service provider class. For local calls no Web protocols are required, which increases the performance. The service manager keeps track of the called service providers to ensure proper transaction handling (see section 4.5).

4.3.2 Core Service Calls from the User Interface

We can separate a service-oriented software solution into two parts: a front end and back end. The front end comprises all user-facing components of the solution; the back end processes the input received from the front end. Front end and back end interact via Web protocols.

The AP / ByDesign user interface (UI) is provided by SAP NetWeaver Portal and Web Dynpro runtime on the Java stack of SAP NetWeaver Application Server, whereas the business logic resides on the ABAP stack (for details on UI technologies see chapter 6). For accessing core services from the user interface, communication between the Java stack and the ABAP stack is necessary. For this, ESI provides two proxies: the generic consumer proxy (GCP) and the remote consumer proxy (RCP).

On the front-end server GCP offers interfaces to Web Dynpro runtime for calling the individual core service operations. For execution, GCP passes core service calls to RCP, which is located in the back end (ABAP stack of SAP NetWeaver Application Server) via SOAP/HTTP. To save roundtrips and speed up communication, GCP can collect multiple core service calls and send them as one request to RCP (see figure 4-3).

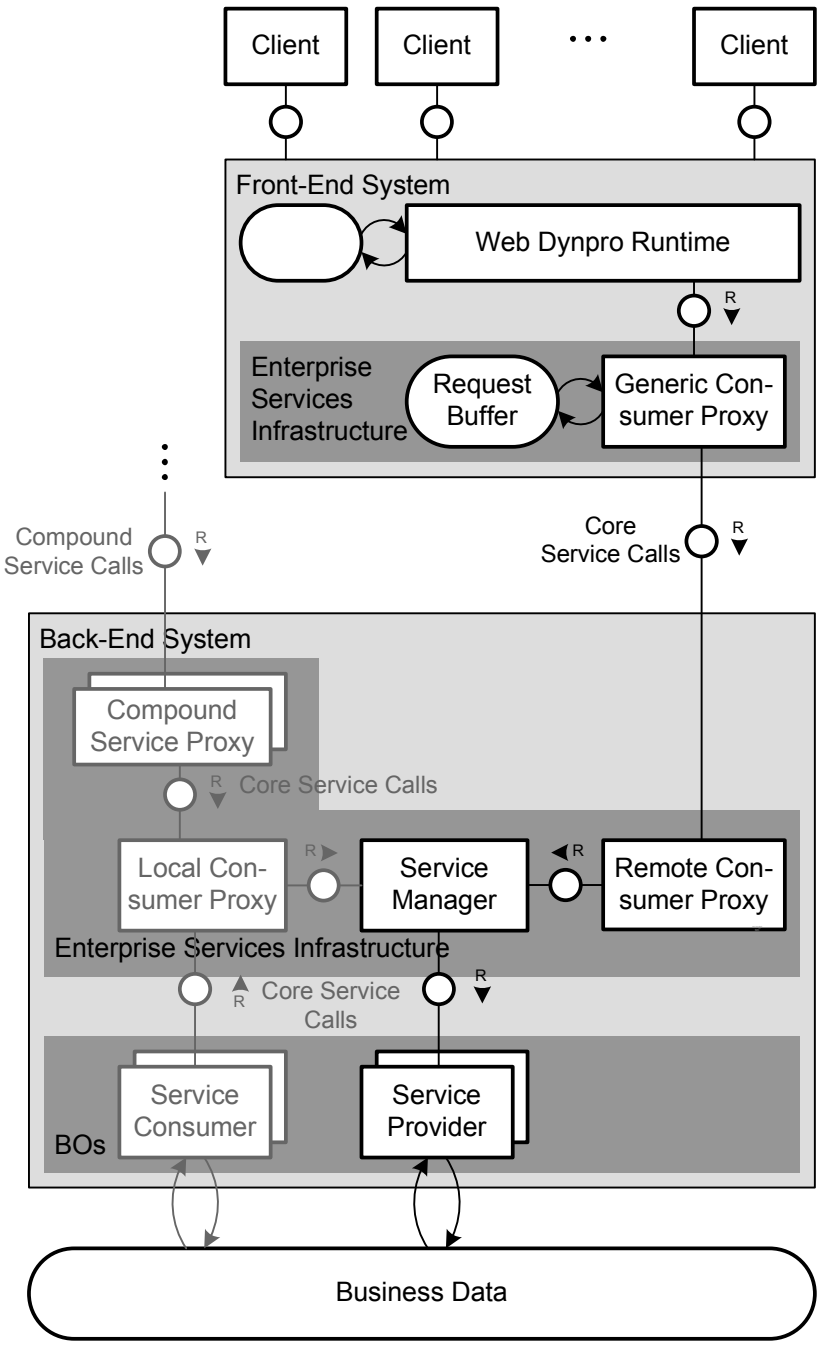


Figure 4-3 Calling Core Services via GCP and RCP

4.3.2.1 Call Sequence

The user's client system communicates with SAP NetWeaver Portal and Web Dynpro runtime via HTTP(s) requests (for details see chapter 6). For each kind of user interaction (query and read data, modify data, execute an action, and so on), Web Dynpro runtime calls the corresponding core service operations at GCP. GCP collects the calls until Web Dynpro runtime signals, with the command `flush`, that the batch of core service calls should be

executed. A batch can contain, for example, multiple retrieve calls for reading data of several nodes of one business object instance that are to be displayed simultaneously on the user interface., Concurrent updates of header and item data of one business object instance are typically transferred together in one batch. Web Dynpro runtime usually sends the command `flush` to the GCP each time the end user finishes an interaction step.

GCP sends the whole batch of calls to RCP, which passes them as single calls to the service manager. The service manager maps the core service operations to the corresponding methods of the service provider classes. In this way the service manager keeps track of the called service provider classes to ensure proper transaction handling (see section 4.5.2). CREATE, UPDATE, and DELETE calls of one business object are collected by the service manager and passed collectively to the referring MODIFY method.

An example of a typical call sequence between Web Dynpro runtime and the service provider is shown in figure 4-4.

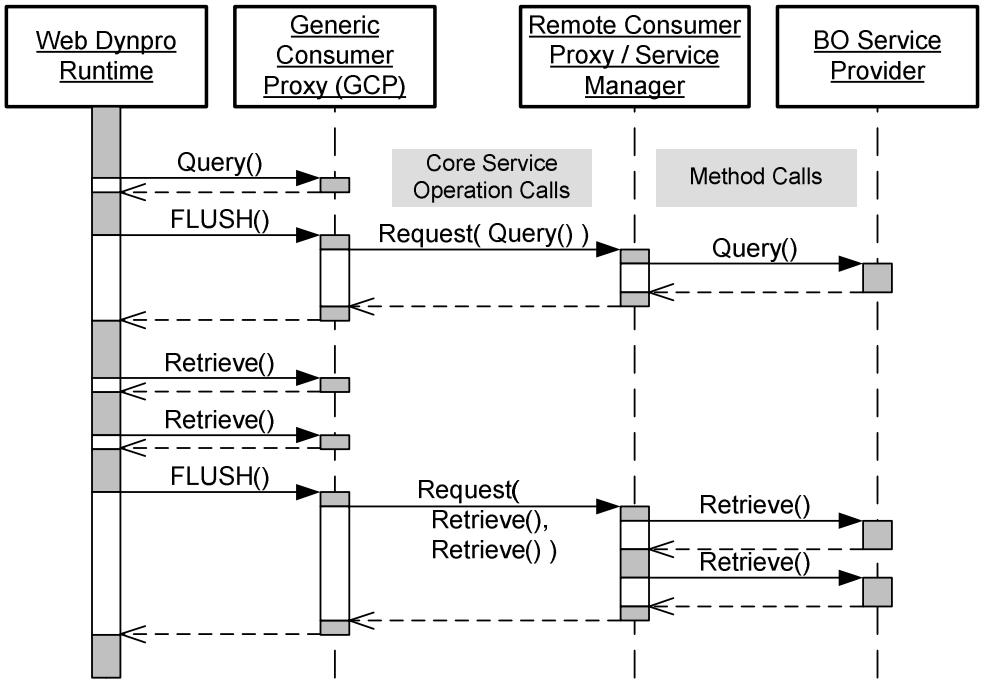


Figure 4-4 Calling Core Services from the User Interface

In figure 4-4 the user interaction starts with a query. The query call is passed to the GCP on the front-end server. Web Dynpro runtime calls the command `flush`, indicating that the query can be executed immediately. GCP sends the query call via RCP to the ser-

vice manager, which calls the QUERY method of the corresponding service provider class. The QUERY method provides the search result as a list of IDs of business object node instances. After getting this response, Web Dynpro runtime calls the RETRIEVE core services of all business object nodes to be displayed on the screen. GCP collects these calls. After the flush from the Web Dynpro runtime, GCP sends them as one request to the back end (RCP). The corresponding data is now read from the database and sent to the front-end server, which displays the data on the user interface.

4.3.3 Events

Up to now we have discussed only the case where communication is initiated by the service consumer requesting a response from the service provider. In contrast, event-driven scenarios follow a publish-subscribe concept, where certain information is pushed to interested parties. In these scenarios, an event is raised when a predefined situation occurs, and all parties registered for it are notified. In the current release ESI recognizes the following events:

- Event BUSINESS OBJECT CHANGED is raised when modified business object node instances are written to the database.
- Execution of core services is successful or leads to errors. In both cases the service consumer needs to be informed via success or error messages.
- Properties of data entry fields change. For example, when an input field is switched to “read only,” this information needs to be published to the user interface.

ESI provides three handlers that deal with these types of events: change handler, message handler, and property handler.

Each service provider class reports each change of a business object node instance to the change handler. The change handler is responsible for publishing these change events to all subscribed parties, which use this information, for example, for triggering data replication or message exchange (see also chapters 5 and 7).

The message handler publishes text messages from the service provider to all subscribers. The messages may include errors, warnings, success, or other process- or business object-related information.

The property handler signals the dynamic properties of business object nodes, attributes, and actions to all subscribers. In ES Repository, properties like READ ONLY or DEFAULT VALUE can be defined for attributes or nodes; ENABLED/DISABLED can be defined for actions. As ES Repository is exclusively a design-time tool, at runtime these properties are stored in the property handler and can be overwritten by the service provider if necessary.

4.3.4 Identity and Access Management

Identity and access management provides central services to manage user accounts and authorizations including registering users, enabling access to business applications, locking users, and removing users from the system. Identity and access management is guided by the following principles:

- The user's UI specifies his authorizations according to the guideline "what you see is what you are allowed to do."
- Authorizations are defined on the level of business objects and core service operations.
- Authorization checks are performed automatically by ESI and need not be implemented by application developers.

Authorizations are checked on the UI level and in the back end. The front-end server ensures that the user sees only those UIs for which the user has permission. The back end ensures that only services and business objects are accessed for which the user has authorizations.

4.3.4.1 The User's Identity

The business object IDENTITY combines all attributes that describe a natural person and all attributes that are required for system access, such as user name and password. This provides a consistent basis for representing the identity of natural persons such as supplier, customer, and employee. IDENTITY is used by all applications and services which have to grant system access or create user accounts.

4.3.4.2 Authorizations and Work Centers

According to the role and position a user hold in the company, he is assigned to work centers, which provide user interfaces to con-

trol certain work areas (see chapter 6.1.1). Work centers are the primary entities for authorization assignment. By default, each user gets authorizations to access all business objects and core services visible in the user's work centers.

For greater control permissions can be restricted to a single operation, action, or business object instance. Organizational and reporting-line units are used when restricting permissions. For example, within the work center SALES ORDER MANAGEMENT a user can access only sales orders owned by SALES OFFICE MINNEAPOLIS.

4.3.4.3 Authorization Checks by ESI

Each business object has an access control list (ACL), which is associated as a dependent object. For each object instance, ACL stores the responsible roles (business functions) and organizational units.

At runtime, ESI automatically performs authorization checks by using access management whenever a core service is called (see figure 4-5). Access management checks the user's authorizations against the ACL of the requested business object instance. If both match, the core service can be executed; otherwise access management rejects the access.

For compound service calls, the remote user first needs to log on to the service provider's system. Access management checks whether the user has the authorization for the requested compound service operation and the related core services.

It is possible to call both, core and compound services, in privileged mode to skip authorization checks. Core services can use privileged mode only for local program-to-program calls via LCP, whereas compound services can be called in privileged mode if sending and receiving deployment unit run on the same system.

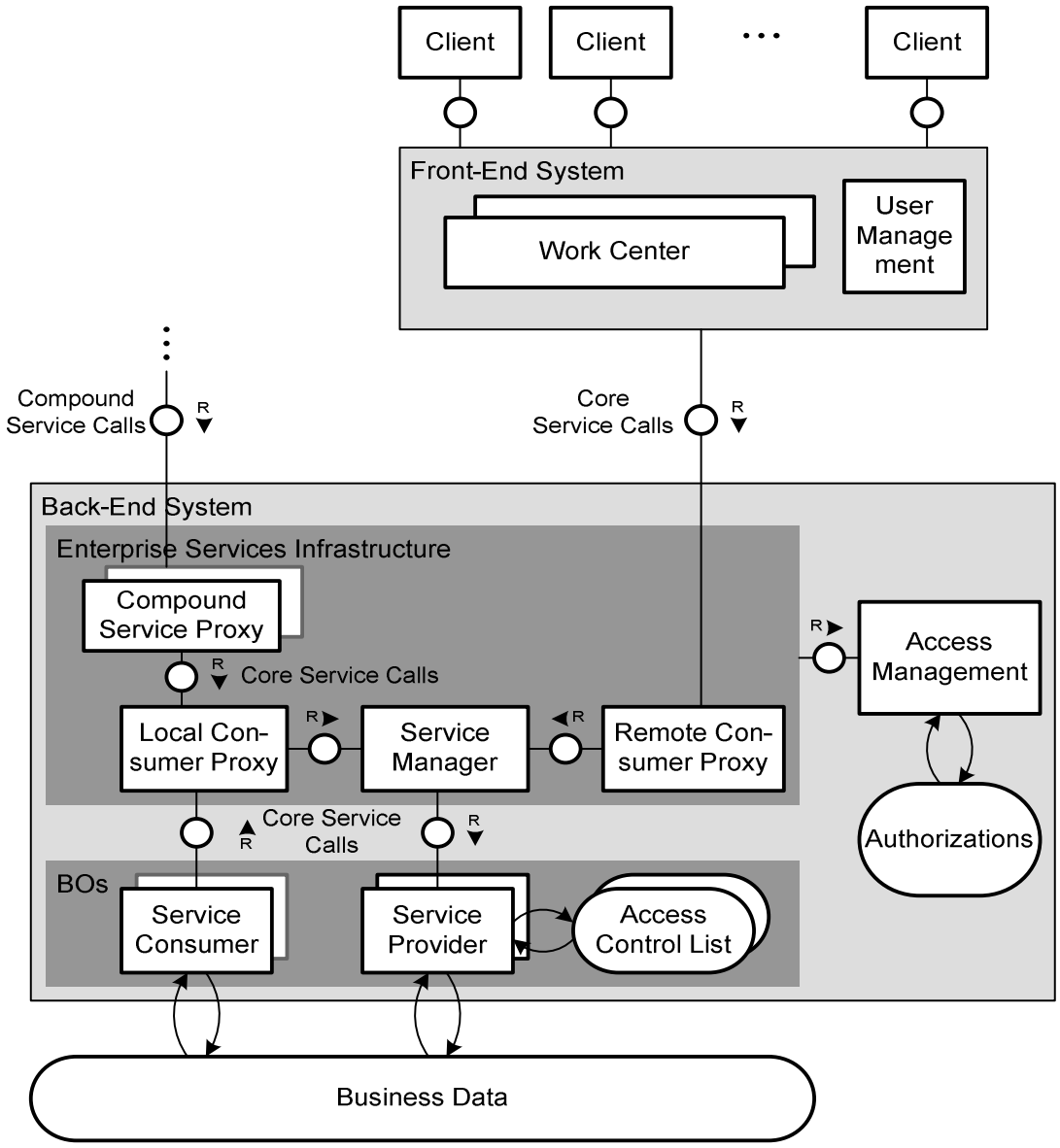


Figure 4-5 Checking Authorizations

4.4 Compound Services

As mentioned in chapter 3.3.2, compound services are Web services that are used in particular for application-to-application (A2A) and business-to-business (B2B) communication. Usually their operations are defined using attributes from multiple business object nodes.

4.4.1 Implementation of Compound Services

Compound service interfaces are designed in ES Repository, too. A proxy class needs to be generated for the implementation of the inbound and outbound interface of a compound service (see chap-

ter 3.3.4.1). The interface of the proxy class provides a method for each operation. Between the compound service proxy and the business object(s), a process agent is implemented as mediator (see chapter 5).

Messages received via a compound service are processed by calling core services via LCP (see figure 4-2).

4.4.2 Execution of Compound Services

Compound services are stateless and can be used in synchronous and asynchronous communication scenarios. For A2A and B2B communication, asynchronous message exchange is preferred (see chapter 5), while the interaction with composite applications (see chapter 8.3.1) is based mainly on synchronous communication.

ESI provides a complete Web service runtime for enabling and managing message-based communication between compound services via SOAP/HTTP. For asynchronous communication, the Web service runtime provides reliable messaging based on persistent queuing and message transport according to the Web service reliable messaging (WSRM) standard. Reliable messaging guarantees that messages are delivered only once and in order. Duplicates are filtered out.

All compound services are published as Web services. As mentioned in chapter 2.3.1, ESI provides an UDDI-based enterprise services registry for configuring, publishing, and discovering concrete compound services in the context of a specific customer installation.

4.5 Managing Transactions

Multiple service consumers can perform concurrent updates of business object instances in main memory. Usually some updates depend on each other and need to be stored in the database simultaneously. Therefore, when the service provider stores the business object data in the database, it must be ensured that the data remains in a consistent state. ESI provides a central transaction management for performing updates in a reliable fashion.

4.5.1 What Is a Transaction?

A transaction advances a system from one consistent state to another. It has an explicit start and end. In conformance with the ACID principle (atomicity, consistency, isolation, and durability), all data changes occurring between the start and the end of a transaction are processed as an atomic unit. Transactions provide a simple model of success or failure. A transaction either commits (that is, all its actions are performed), or it aborts (that is, all its actions are undone or rolled back). This all-or-nothing quality makes for a simple programming model.

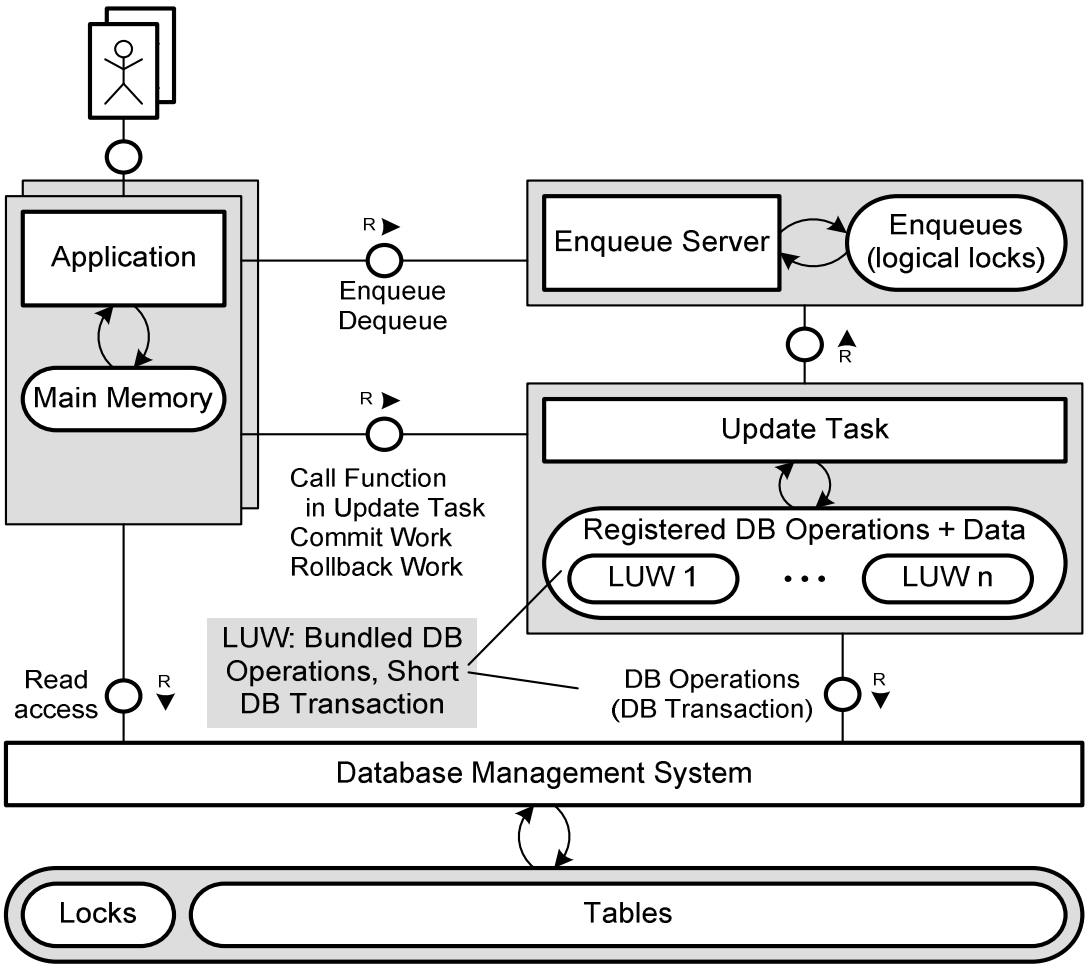


Figure 4-6 Applications Using Enqueue Server and Update Task

On the database level, transactions are handled by database management systems, which provide different levels of isolation with different locking strategies. In the context of business applications, only a few scenarios need a strict level of isolation to ensure data integrity; the majority just needs fast read-only access. Auto-

matic locks on the database level may be problematic for many business applications. The specific requirements of business applications, together with technical restrictions, led to the SAP enqueue server/update task concept where transaction handling and locks are handled outside the database management system.

While database transactions should be as short as possible to ensure minimal database locking intervals, transactions on application level may span long, undetermined time intervals, because a consistent transition of business object data normally takes several user interactions. SAP uses the term logical unit of work (LUW) for transactions on an application level.

An LUW is a collection of database operations. Within one LUW, the application operates on data in main memory and registers functions for the update task (`CALL FUNCTION IN UPDATE TASK`), which includes database operations and mapping data from memory to database. When committing the LUW (`COMMIT WORK`), the update task performs all registered database operations in one short database transaction.

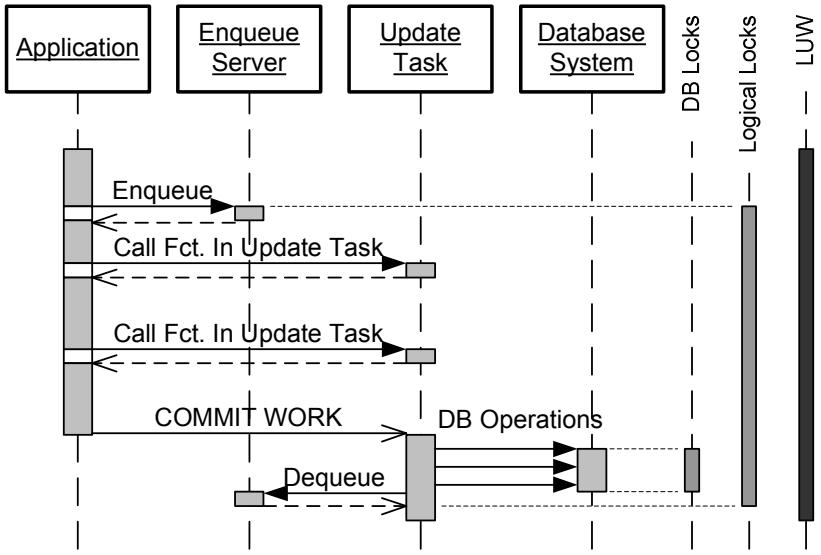


Figure 4-7 How Applications Use Enqueues and Update Task

Business data that is being updated by a user is locked on an application level using enqueues to prevent other users from changing the same data. Enqueues are logical locks for application-specific entities; they can cover anything from single nodes up to complete objects and are not tied to entities in the database such as table rows. Enqueues have to be checked only by applications

that require the latest and most consistent view or need to modify data. This approach scales better than using the locking mechanisms as provided by the database management system.

In AP / ByDesign LUWs are restricted to one deployment unit, because deployment units need to be loosely coupled and separately operable. Processes interacting with multiple deployment units require multiple LUWs. Each compound service is executed in one LUW due to its self-contained nature.

During the LUW, all changes and updates are kept in main memory by the service providers across multiple core service calls. When the service consumer commits or aborts the interaction, ESI signals the end of the LUW to all service providers. All changes on the service provider side are persisted if the consumer finishes the interaction; all changes are discarded if the consumer aborts the interaction. The detailed processing of a LUW is described in the transaction schema below.

4.5.2 Transaction Schema

To allow ESI to control and manage the LUW, each service provider class has to implement a set of methods for transaction handling, such as readiness checking, triggering database operations, and cleaning up after the successful end of the LUW or failed readiness checks. These methods are combined within the transaction interface (see chapter 3.3.3.4). During a LUW they are called by ESI in a predefined order according to the following transaction schema (see figure 4-8):

Initialization – Begin of LUW

A new LUW is started either by user interaction (core service call) or by an incoming message (compound service call). After ESI has created the transaction ID for the LUW, the requested service provider classes are instantiated and their INIT methods are called. Within those methods, service providers prepare for handling service requests, for instance by reading configuration tables. If required, they subscribe for change events of other business objects.

At the end of the INIT operations the service providers are ready for interaction.

Interaction Phase

During the interaction phase the service consumer calls core service operations such as QUERY, RETRIEVE, UPDATE, and ACTION. Service providers track all changes of their business object instances. They are also responsible for setting enqueues on business object node instances that are requested for modification. The finest granularity of an enqueue is a business object node; the coarsest is the business object.

Save Phase

When the core service operation SAVE is called, ESI loops all instantiated service provider classes and calls the following methods:

- Finalize
  The call of this method signals the service provider that the transaction is now in its final state. Within the finalize method the service provider performs final processing of business object data to ensure that all its business object instances contain consistent data. After final processing the business object instances cannot be changed by further core service calls, neither from RCP nor from LCP, during the running transaction.
- Check_Before_Save
  Within this method the service provider checks data consistency and decides whether all changes as a whole can be written to the database. If not, the save phase is stopped and error handling is started.
- Save
  Within the SAVE method the service provider assembles all database operations required by the changes and sends them to the update task via `call function in update task.`

Immediately before ESI calls the SAVE method and directly after the SAVE method has been processed, it publishes BUSINESS OBJECT CHANGED events to registered parties. Process agents that initiate asynchronous message-based communication are registered to events published at BEFORE SAVE (see chapter 5.3.1). Business intelligence agents that replicate data for analytics are registered to events published AFTER SAVE (see chapter 7.1.4.2).

Commit Work

After all instantiated service providers have successfully processed their SAVE method, ESI closes the LUW by calling `commit work`. By executing registered database operations, the update task now writes the data to the database within one short database transaction. If an error occurs during the SAVE processing, the transaction is aborted and ESI initiates a rollback of the LUW.

End of LUW

All enqueues are removed, and the LUW is finished.

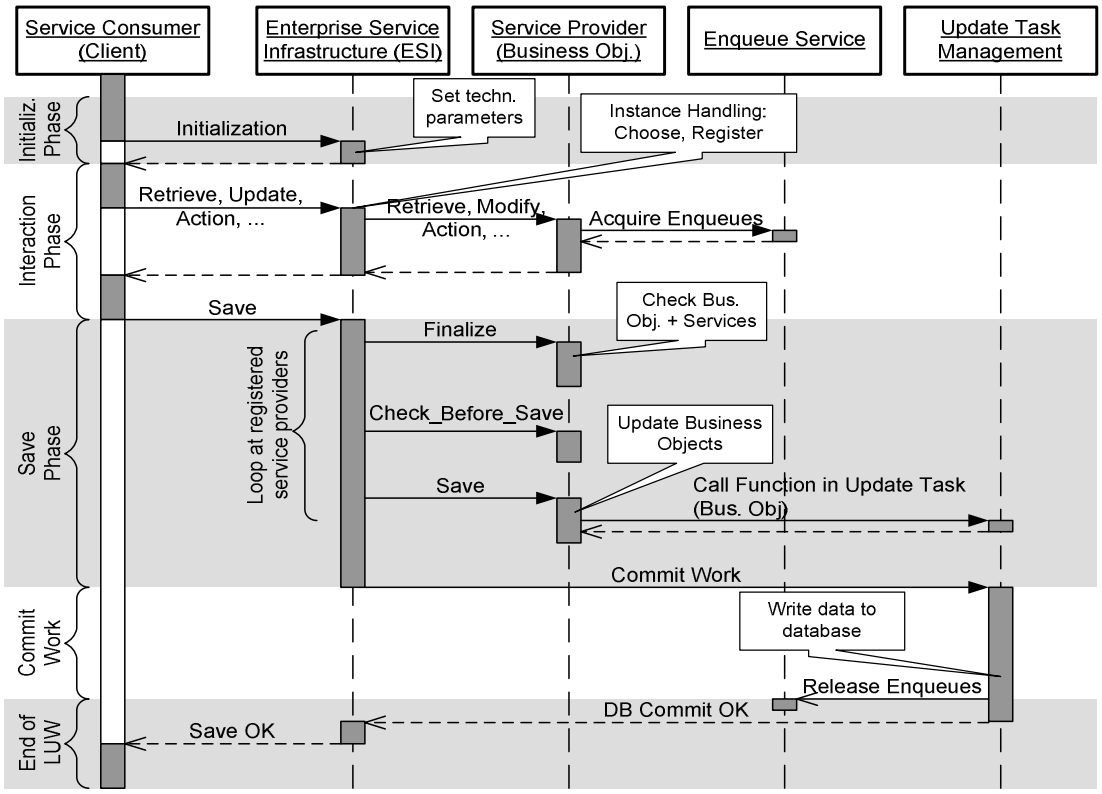


Figure 4-8 Transaction Schema

4.5.3 Consequences for Business Object Implementation

The following consequences, derived from the transaction schema explained above, apply for implementing a service provider class (business object):

- The ABAP programming language contains statements for initializing, committing, and aborting LUWs, such as `COMMIT WORK`, `ROLLBACK WORK`, `SET UPDATE TASK`

`LOCAL`, `PERFORM ON COMMIT`, `PERFORM ON ROLLBACK`, and `MESSAGE`. Each ABAP statement that affects transaction handling must not be used for implementing core service operations, because the transaction is exclusively controlled by ESI.

- The methods required by the core service interface pattern TRANSACTION (see chapter 3.3.3.4) have to be implemented within each service provider class.
- Each service provider has to track all data changes. Within the save method the service provider assembles the corresponding database operations and passes them to the update task via `call function in update task.` Direct database updates are not allowed.
- The service provider has to set proper enqueues for data requested for change. Enqueues are released automatically during the commit phase.

The rules described are mandatory for all development based on ESI.

4.6 In a Nutshell

Enterprise services infrastructure provides design-time tools and runtime engines for creating, providing, and invoking services. Developers use the central enterprise service repository to define business objects, compound service interfaces, and data types. According to the defined metadata, proxies and skeleton classes are generated.

ESI provides several generic proxies for calling core services: the local consumer proxy is used for local calls from a business object or compound service. Generic consumer proxy and remote consumer proxy enable communication between the user interface and business objects. If a core service is called, ESI automatically checks the necessary authorizations using identity and access management.

Core services are stateful services that support synchronous communication only. Multiple core services can be called in a specific sequence as one logical unit of work in order to get a meaningful result and to maintain data consistency. Transaction handling is done by ESI.

Compound services are stateless and can communicate asynchronously as well as synchronously. One compound service call is always performed within one LUW. ESI provides a complete Web service runtime for enabling and managing message-based communication between compound services via SOAP/HTTP.

5 Using Message-Based Process Integration

End-to-end business processes go across different applications or process components within or beyond enterprise boundaries. This is what we refer to as business process integration. In particular, enterprises need support for the following integration scenarios:

- Business-to-business communication (B2B)
  The exchange of business documents between enterprises that follows a specific message choreography, usually in conformance with e-business standards like EDIFACT and RosettaNet
- Business process outsourcing (BPO)
  The operation of a certain application or process component, for example, human resource management, by an external provider
- Integration of third-party systems
  The interaction of SAP applications with third-party applications
- Deployment on different systems
  The distribution of process components among several systems to support a combination of centralized and decentralized business operations

In AP / ByDesign, these scenarios are realized using loosely coupled process components that reside in different deployment units and communicate through asynchronous message exchange. These process components are black boxes for other components with their business objects accessible only via published compound service interfaces. Communication through dedicated interfaces simplifies process modeling and makes the process flow transparent.

The process integration scenario SELL FROM STOCK introduced in chapter 3.1 relies, for example, on the process components SALES ORDER PROCESSING, CUSTOMER INVOICE PROCESSING, and CUSTOMER REQUIREMENTS PROCESSING. Each of these three process components runs in a separate deployment unit. Although we are focusing in this book solely on AP / ByDesign, process compo-

nents could also be part of the SAP Business Suite or located in applications from other software vendors.

In this chapter we explain how process integration between loosely coupled process components is achieved using compound services and process agents.

5.1 Message-Based Process Integration

In general, the communication medium influences how the interaction takes place. Either communication partners talk directly to each other, or they exchange messages containing business documents.

- Call-based communication
  Call-based communication requires an established communication channel between two peers who both must be online. The typical example for this type of communication is a phone call.
- Message-based communication
  In a message-based communication scenario, peers address and send messages to each other. No direct online communication channel needs to be established. A typical example is an (e-)mail service.

Both call-based and message-based communication can be performed in a synchronous or asynchronous way:

- Synchronous
  A client sending a request to a server synchronously is blocked from further operation until the server sends back the result.
- Asynchronous
  A client sending a request to a server asynchronously is not blocked from further operation after sending the request, but must provide a way to receive the response from the server later on.

Synchronous, call-based communication is the default for subroutine and intra-process calls, as sender and receiver need to be tightly coupled. On the other hand, if sender and receiver should be loosely coupled, for example for communication across process or system boundaries, then asynchronous, message-based communication fits best.

In AP / ByDesign, both communication types are used:

- Within a deployment unit, business objects and process components are tightly integrated using the common database. They communicate synchronously and call-based via core services. In this case, service consumer and service provider run in the same logical unit of work (LUW).
- In contrast, process components of different deployment units are loosely coupled. Service consumer (sending process component) and service provider (receiving process component) are executed within separate LUWs. They communicate via their compound service interfaces using asynchronous message exchange. Synchronous, message-based communication across deployment units is used only in exceptional cases when the business scenario enforces a direct reply of the receiving process component. (For more details on deployment units, see chapter 2.3.4.1.)

The loose coupling of process components residing in different deployment units makes it easier to assemble business processes from different process components and to support various deployment scenarios. Several deployment units can be operated together on one system sharing one database or in a distributed approach, where each deployment unit runs on a dedicated system with a separate database. In the distributed approach, each deployment unit has its own foundation providing reuse components and master data such as customer, material, and organizational management information. Master data is aligned between the different systems using data synchronization services provided by the technology platform.

5.1.1 Central Versus Decentral Process Integration

Messaging between process components is initiated by events from the sending process component (see chapter 4.3.3). In general, the message exchange includes the following steps:

- Evaluation of whether a message needs to be sent (according to the business process definition)
- Determination of the receiver of the message (responsibility determination)
- Mapping of message data to the interface of the receiver

- Routing of the message to the receiver (routing to technical destination)
- Checking the correctness and consistency of the received message

From a principle architecture perspective, these steps could be executed centrally by an integration hub or decentrally within the process components itself. Using central process integration would mean that each event has to be forwarded to the central integration hub to decide whether to send a message to initiate a subsequent process step. In AP / ByDesign, this decision is usually based on context information of the process component, such as status, changes to business object data (before and after images), and business configuration data. Forwarding all this information to a central hub would cause additional effort and performance degradation.

Instead, decentralized integration logic can access business object data much faster via main memory. The same is true for receiver determination and the correctness check of the received message, which both need specific information from the business objects. Therefore, in AP / ByDesign, all of these steps are executed decentrally by process agents, which are controlled by the process agent framework. If additional mapping and routing is required, notably for B2B communication or third-party integration, this is handled centrally by SAP NetWeaver® Exchange Infrastructure (SAP NetWeaver XI). A detailed description of SAP NetWeaver XI is available in [NFN+06].

Although process integration logic is executed decentrally in AP / ByDesign, design, configuration, and monitoring of process integration is centralized. The message choreography between process components is modeled centrally in the enterprise services repository (ES Repository). The concrete business process implementation, including process agents, is derived from these models. During business configuration in the central design and scope definition system (see chapter 8.1), the message exchange is configured and the corresponding process agents are activated according to the selected business processes.

5.1.2 Process Agent Framework

In general, agents are program entities that act partially autonomously at runtime in order to accomplish certain tasks or make decisions. Currently AP / ByDesign distinguishes the following types of agents, which are triggered and executed by the process agent framework:

- Process agents, which assemble and process messages sent or received via compound service interfaces for synchronous as well as for asynchronous message exchange
- Business intelligence agents (BI agents), which extract business object data for analytics (see chapter 7.1.4)
- Task agent which initiates business tasks (for processes that require certain user actions, see section 5.4.3)

Agents react to specific events. For example, process agents and BI agents subsribe to events indicating changes of business object instances. The task agent is called to trigger a certain user activity, for instance, when an error occurs. When triggered by an event, agents perform a predefined sequence of steps (see chapter 5.2) in order to fulfill their task.

In AP / ByDesign, agents are the key facilitators for message-based process integration. They run under control of the process agent framework, which triggers process agents, monitors process execution, and provides process history information. The process agent framework as runtime infrastructure embraces the following main elements for message-based communication (see figure 5-1):

- Process agent controller
- Process integration persistence
- Error and conflict handling

The process agent controller triggers process agents and ensures that the steps of a process agent are executed in the predefined order. In addition, the controller gives process agents access to the process integration persistence. If an error occurs at process agent execution, the process agent controller can initiate a rollback of the running transaction to ensure data consistency.

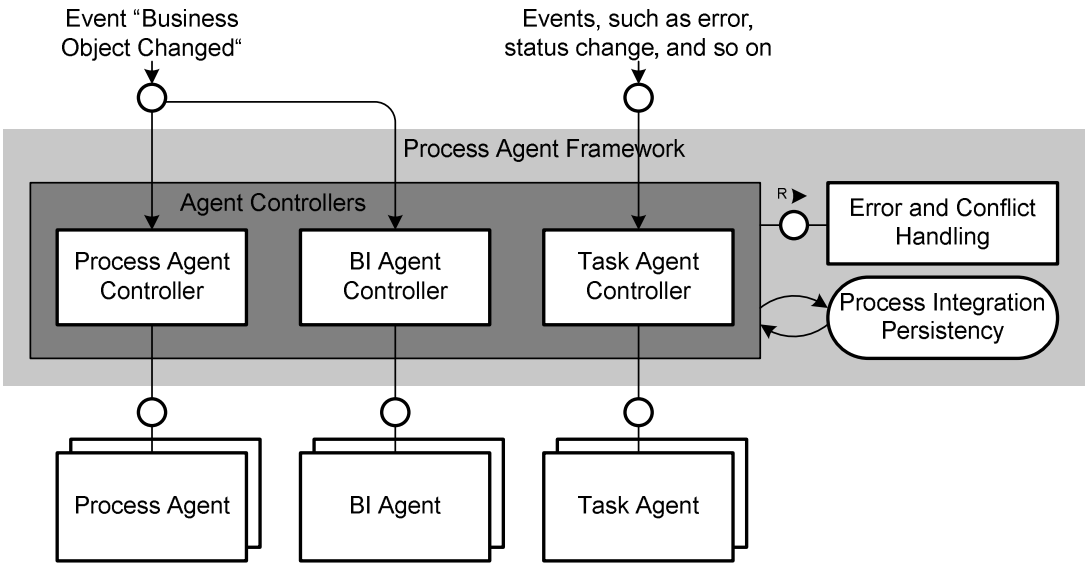


Figure 5-1 Process Agent Framework

Process integration persistence stores information about sent and received messages, and references between message content and the corresponding business object nodes. It provides appropriate service interfaces to access this information, which is used to support decision making of the process agents, sequencing of messages, and process monitoring.

In contrast to synchronous, message-based communication, asynchronous message exchange always requires error and conflict handling (see chapter 5.4) on the receiving side, because no direct feedback channel to the sender can be anticipated. If a process agent detects an error or conflict when receiving and processing a message, error and conflict handling is called. Based on the analysis of the process agent, error handling initiates manual or automatic follow-up steps to solve the problem.

5.2 Process Agents

Process agents encapsulate process integration logic. They act as mediators between business objects and compound service interfaces. The process agent framework supports two types of process agents: Outbound process agents and inbound process agents.

- Outbound process agents
  Evaluate whether subsequent process steps have to be initiated in another process component. If yes, they collect the

necessary data and assemble the message to be sent via the compound service interface (outbound interface) to the receiving process component. In the case of synchronous communication, the outbound process agent waits for the reply.

- Inbound process agents
  Receive messages from the compound service interface (inbound interface). They process the message by calling the affected core services of the corresponding business object nodes. In the case of synchronous communication, inbound process agents assemble and send the reply message back to the service consumer.

Each outbound process agent is registered to the events of exactly one business object and sends messages via exactly one outbound interface. Therefore, a business object can have several outbound process agents, each one triggering a subsequent business process in another process component.

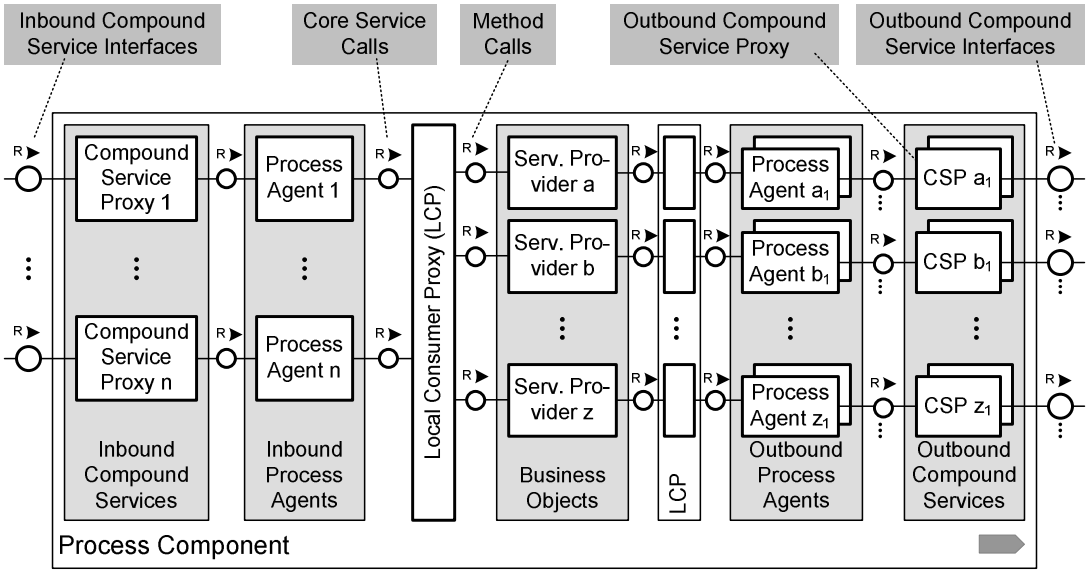


Figure 5-2 Relationships of Compound Services, Process Agents, and Business Objects

An inbound process agent is triggered by messages received from exactly one inbound interface. One inbound process agent can update several business objects in parallel (see figure 5-2).

The interacting outbound and inbound process agents implement the message choreography, meaning the sequence of sent and

received messages necessary for a complete conversation. Message choreography together with the involved business objects, process agents, and service interfaces is modeled in ES Repository (see chapter 5.1.1). Based on these models, the required proxy classes for compound service interfaces and the code skeleton for process agent classes are generated in the local development environment:

- The compound service proxy provides a callable interface.
- The skeleton of a process agent class provides a frame for implementing the process interaction logic.

The process integration logic provided in a process agent is divided into a number of steps that are executed sequentially (see chapter 5.3.1 and 5.3.2). Each step to be executed by an inbound or outbound process agent is implemented in a separate method of the proxy class.

With the introduction of process agents, process integration logic in AP / ByDesign is clearly separated from business logic implemented in service provider classes (business objects). With this approach, process integration can be developed and maintained more easily. For example, new processes can be introduced just by adding new agents.

5.3 Asynchronous Communication

Asynchronous communication is always preferred for integration of decentralized process components because of loose coupling. With asynchronous communication, the sender does not need to wait until the receiver of the message responds, but can continue processing even if the receiver is not available.

Asynchronous communication is explained in this section. However, there are use cases requiring synchronous communication, which is described in section 5.5.

5.3.1 Outbound Processing

Each outbound process agent is registered for the change events of its corresponding business object. These are logged by the change handler of the enterprise services infrastructure (ESI). After triggered by an event, outbound process agents decide, based

on context information, whether subsequent business process steps have to be initiated, and send messages if required.

As explained in chapter 4.5, a new LUW is started either by user interaction (core service call) or by an incoming message (compound service call). During the LUW, the change handler of ESI records all changes to business object node instances. As soon as the core service SAVE is called from the user interface or from an inbound process agent, the change handler publishes the change events to all subscribed parties.

Each process agent is registered to the change events of exactly one business object. The process agent framework triggers all outbound process agents registered to a specific event. After receiving the event, the outbound process agent executes the predefined steps (see below). The process agent controller triggers the execution of the steps in the right sequence and monitors the result.

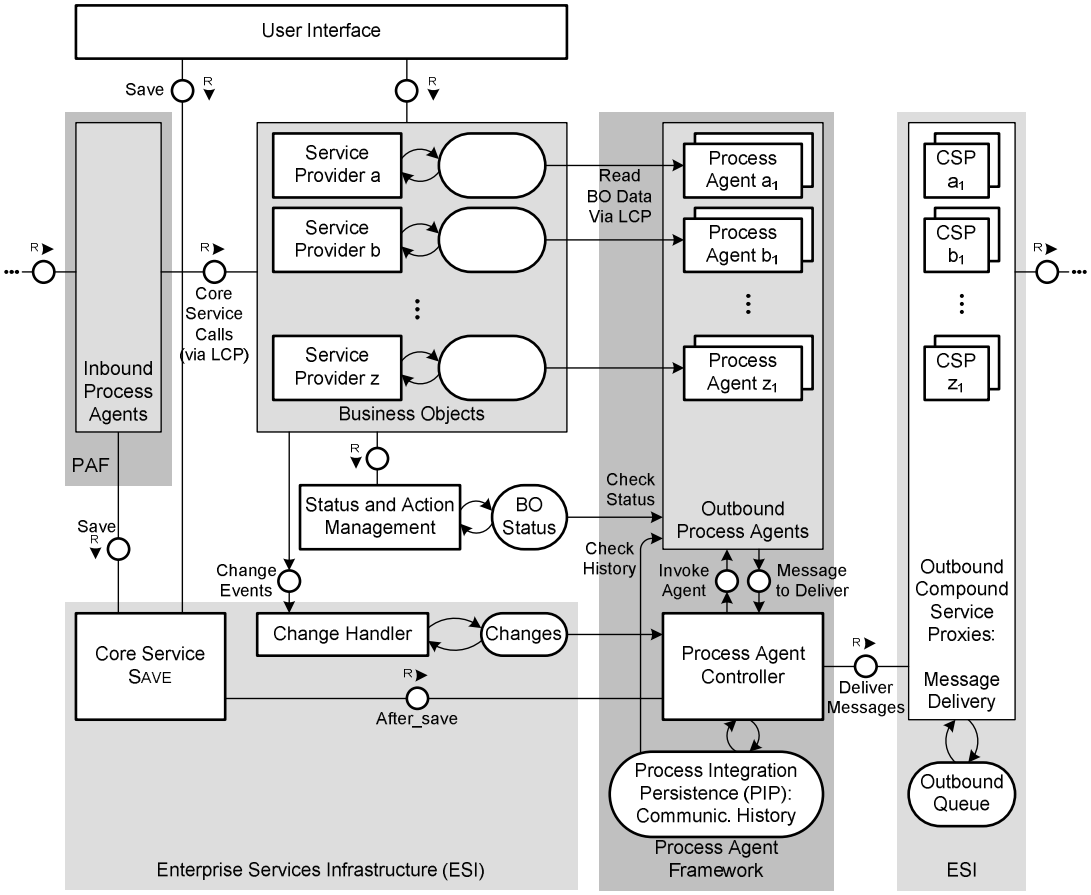


Figure 5-3 Architecture for Outbound Processing

After being triggered by the process agent framework, an outbound process agent first decides whether a message needs to be sent at all. If yes, the agent determines the message receiver and prepares the message content. Of particular importance to the decision is the business object's status and process history information stored in process integration persistence. At the start of an LUW, the service provider class stores a copy of each business object instance in main memory (before image). In a second copy, the service provider class performs all updates (after image). The process agent has access to the business object instances' before and after images.

In detail, the outbound agent executes the following steps:

1. Determine process relevance
   In this first step, the outbound process agent decides if the updates of the business object nodes are relevant for the initiation of a subsequent process step. Comparing the before and after image of the business object node instances, the process agent determines which data has been changed and if the changes are relevant for sending a message. To do so, the process agent checks for each changed business object node whether the process relevance condition is true.

   Examples for process relevance conditions are:
   - All sales order items are relevant for billing.
   - All sales order items with the MATERIAL product category are relevant for integration with OUTBOUND DELIVERY PROCESSING.

   For each business object node that complies with the process relevance condition, the process agent executes at least the next two steps.
2. Evaluate process integration persistence
   The outbound process agent determines whether the subsequent process has already been triggered before (for example to avoid duplicates). For that, the process agent checks the process integration persistence, a storage where references to transmitted business object nodes are kept.
3. Condition evaluation
   In this step, the outbound process agent evaluates start, change, or cancel conditions together with the communica-

tion history information retrieved in step 2 to determine what the receiver should do with the data received in the message. Typically start and cancel conditions refer to the status attributes of business object nodes whereas change conditions are based on other dynamic business object data.

The condition evaluation results in action codes, which indicate the actions the receiver of the message should perform. The available action codes are CREATE, CHANGE, DELETE, and NO ACTION. The definition of action codes makes communication more reliable, as certain dependencies between messages are made visible. As an example, consider the loss of a message that contains a business object node instance to be created. In this case, an update of the same node instance is sent within a new message. If action codes are used, the receiver gets the message with action code CHANGE or DELETE, which indicates that the same node instance with the action code CREATE has to be received before. As this did not happen, the receiver detects an error. If action codes are not used, the receiver interprets the update message as a create message, which leads to an inconsistent state.

Only if one or more business object nodes have an action code CREATE, CHANGE, or DELETE, the process agent continues with the next step.

Examples for Condition Evaluation

A start condition is true if an item has the status APPROVED, whereas a cancel condition is true if an already approved item has the status REJECTED.

A change condition is true if the quantity of an ordered product has been changed.

Process history information is required in case of action code DELETE to verify whether a CREATE of a business object node was sent during a previous message exchange.

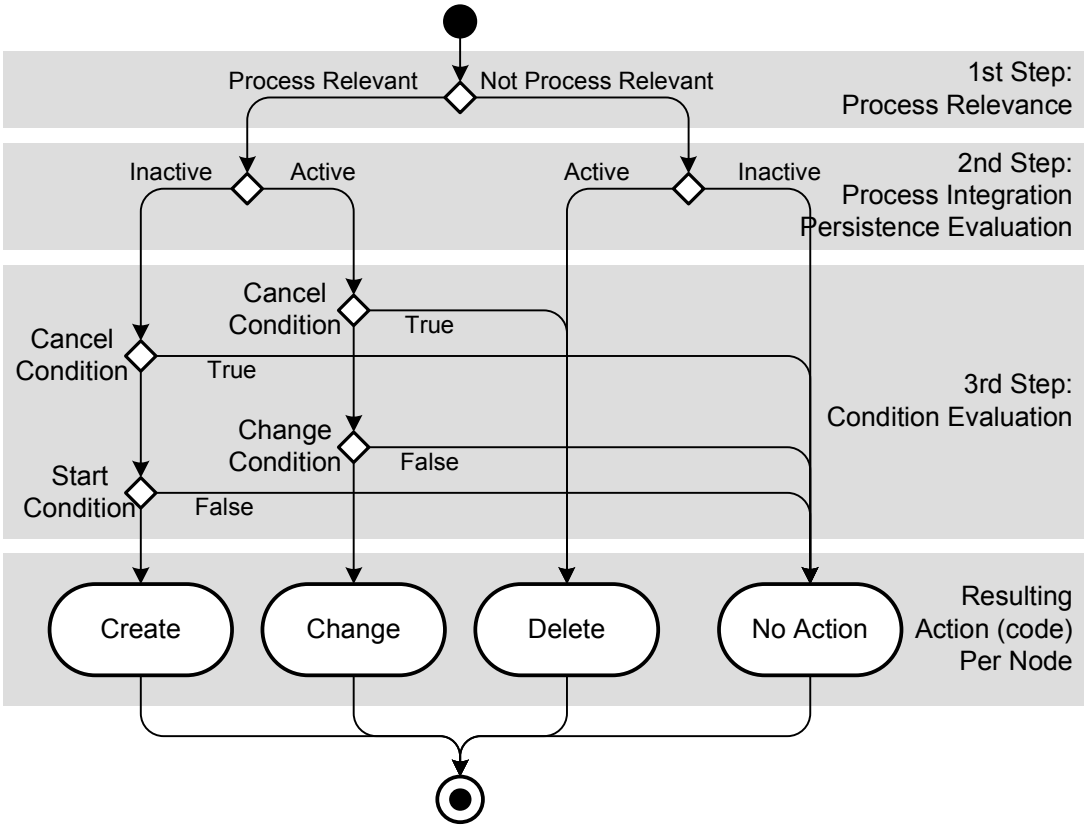


Figure 5-4 Example of Action Code Determination

4. Determine responsibility
   The outbound process agent determines the receiver of the message on the business level. For application-to-application (A2A) communication, the receiver is usually the responsible organizational unit within the company. For business-to-business (B2B) process integration, it is an external business partner.
5. Assemble message
   In the last step, the outbound process agent assembles the message. Accessing business object data in main memory, the process agent creates the message according to the corresponding outbound operation.

   If certain business partners require communication via media other than XML messages – for example, as a printed document, e-mail, or fax – outbound process agents are capable of calling the needed output service (see section 5.3.3).

Finally, the outbound process agent controller takes the assembled message from the process agent and passes it to the

corresponding compound service proxy. After that, ESI finishes the LUW. Business object data changes and the corresponding outbound messages are now stored within the same LUW: business object data changes in the database and messages in the outbound queue of ESI. This ensures transactional integrity.

If an error occurs during execution of an outbound process agent, the outbound process agent controller initiates a rollback of the entire LUW including all outbound process agents.

5.3.2 Inbound Processing

A message created by an outbound process agent is subsequently transferred to the receiving deployment unit using SOAP (see chapter 4.4). There the message is delivered to the proxy of the corresponding inbound compound service operation. A new LUW is initiated by calling the operation (see chapter 4.5).

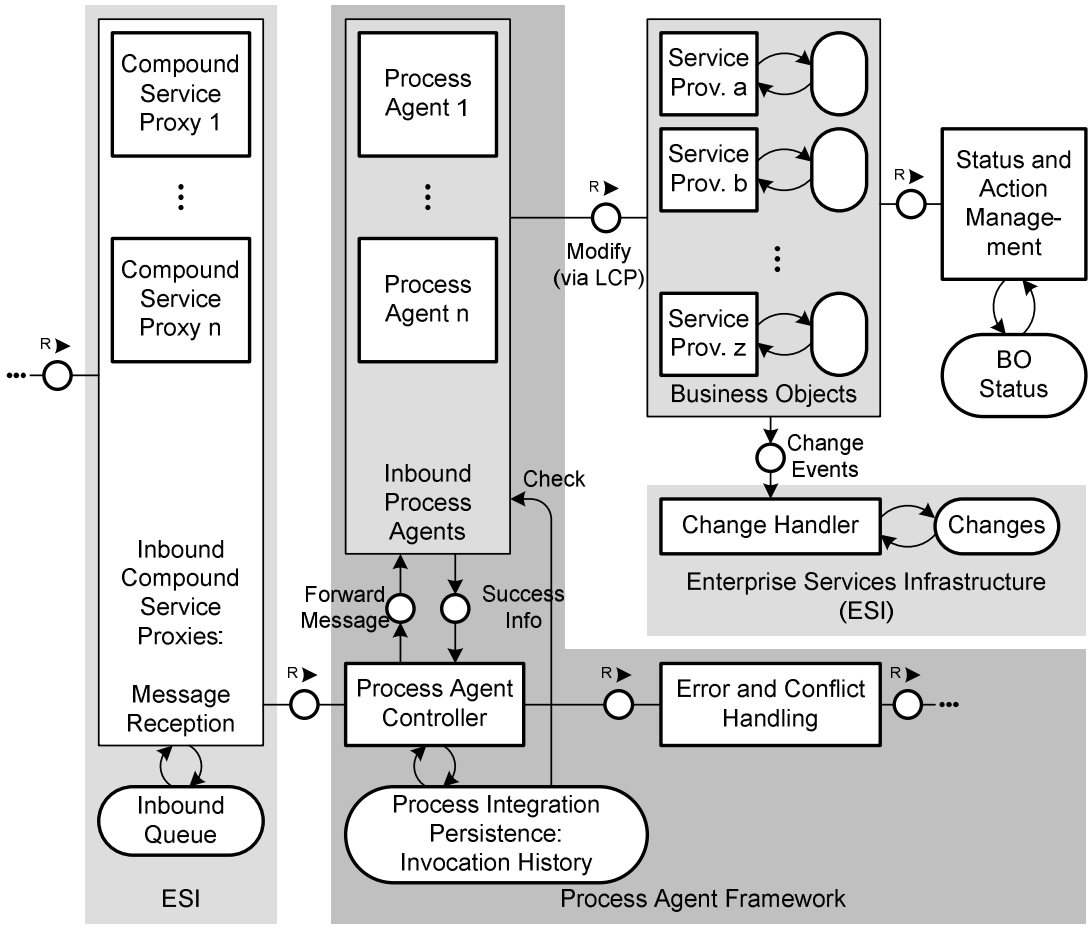


Figure 5-5 Architecture for Inbound Processing

The process agent controller forwards the message to the inbound process agent that is assigned to the requested operation.

The inbound process agent controller calls the steps of the inbound process agent, described below, in the predefined sequence. The goal of the inbound process agent is to update the relevant business objects based on the received message. To ensure a consistent, correct, and quick update, the inbound process agent executes the following steps after receiving the message. Each step is implemented as one method in the inbound process agent class:

1. Set enqueues for business object instances
   This step applies only to synchronous read access. To prevent concurrent updates, the inbound process agent acquires enqueues for all business object instances from which it will retrieve data. Of course, enqueues for modifying business object instances are set by the service provider and not by the process agent.
2. Check message content
   In this step, the inbound process agent checks the message structure and the data format of the incoming message for correctness. The agent verifies, for example, if the referred ISO code describing abbreviated country- or currency-related information is known in the business object's code lists.

   The agent also checks whether business object nodes that correspond to the message content exist, and – based on the process integration persistence – whether the process has already been initiated. During the latter check, the inbound process agent recognizes outdated messages, and as a result, no business object is created, updated, or deleted by an obsolete message.
3. Update business object
   The inbound process agent initiates the update of the business object instances referred to by the incoming message. The business objects and operations related to a given process agent are defined at design time. At runtime, the inbound process agent identifies the requested business object instances based on the message content. The inbound process agent maps the content of the received message to the data structures of the business object's

core service interfaces. According to the action codes included in the message, the corresponding business object nodes are created, updated, or deleted. The core services are called via local consumer proxy (LCP) (see chapter 4.3).

4. Notify of modifications
   The inbound process agent informs the process agent controller about which business object nodes were updated when the received message was processed to ensure correct updates of the process integration history. The business objects themselves notify the ESI change handler about changes.

After the execution of all steps, the ESI closes the LUW by calling the core service SAVE of all participating business objects.

During inbound processing, errors can occur during each of the steps described above, starting with the first check of the business document content right up to saving the business object. In this case, the inbound process agent triggers the execution of error handling (see section 5.4).

5.3.3 Output Management

Exchanging business documents electronically using messages and compound service calls is convenient, but not feasible in all cases. The receiver of the business document may prefer other means of communication like e-mail, fax, or letter, or just does not support B2B communication. For these cases, output management handles the creation of business documents for the output channels of printing, e-mail, fax, and SMS. It supports front-end and back-end output scenarios as well as mass output.

Business documents are created based on forms, which are part of the SAP product delivery or are individually modeled using the forms design tool. At runtime, the output service handles the output requests, calling the form service to select the appropriate form template and fill it with business data. Afterwards the output service transfers the form to the requested output channel (see figure 5-6).

For front-end output, the documents are created in portable document format (PDF), which is displayed at the client using Adobe®

Acrobat Reader. Users can print the document, fax it, or send it as e-mail, using the corresponding features of their client system.

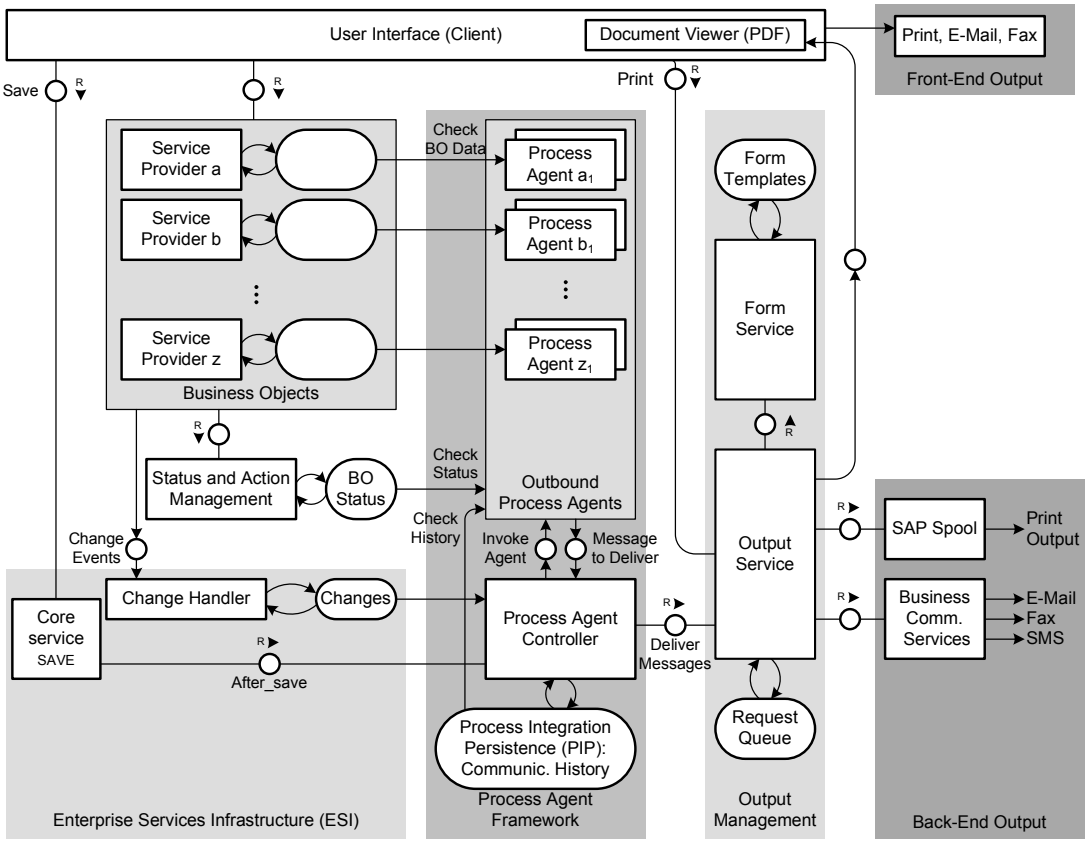


Figure 5-6 Output Management

For back-end output, the output service is either triggered automatically by outbound process agents or manually by a user. The outbound process agent determines the preferred communication channel of the receiver. If this is letter, fax, e-mail, or SMS, the outbound agent calls the output service to create the corresponding document and to trigger the output. Output initiated via process agents is always recorded in the process integration persistence, which allows for business process monitoring later on.

For print output, the form service generates a printable data stream formatted for the selected printer, such as postscript. For e-mail, fax, or SMS communication, a PDF document is created. The output service stores the data stream or document in a queue. Requests are processed immediately one after the other or on a scheduled basis, according to the corresponding configuration settings. For printing, SAP spool interface is used; whereas e-mail, fax, and SMS is sent via SAP business communication services.

5.3.4 Transactions and Data Consistency Across Deployment Units

Business objects and process components within one deployment unit exchange data via synchronous core service calls (see section 5.1). In this case, service consumer and service provider run in the same LUW, ensuring that data changes are saved consistently within the same database transaction (see chapter 4.5). Process components residing in different deployment units usually communicate via asynchronous message exchange. Because the transaction schema, discussed in chapter 4.5.2, is restricted to one deployment unit, it is not sufficient for cross-deployment unit data consistency. In asynchronous communication, data stored by sender and receiver can become inconsistent if messages are processed more than once, get lost, or arrive in the wrong order. When that happens, sender and receiver need to be synchronized.

To ensure consistent data in asynchronous message exchange, the following aspects need to be addressed:

- The business data of a business object instance and of corresponding outgoing messages have to be equal. Only consistent data is transmitted via messages.
- Messaging between sender and receiver must be reliable

The transaction management of the sending deployment unit has to ensure that both the business object data and the messages needed for message-based integration are saved in a consistent way. Across deployment units, specific communication patterns are defined as contracts between sender and receiver (see chapter 3.3.1 and section 5.3.4.2). In addition, ESI provides techniques for reliable messaging and for data synchronization between distributed systems. The latter is called reconciliation.

5.3.4.1 Transaction Handling and Messages

To ensure that the business data sent via messages is consistent with the data stored in the business object instance, the transaction schema is enhanced by the phase BEFORE SAVE (see chapter 4.5.2). BEFORE SAVE occurs directly before the SAVE method of the service provider is called. At BEFORE SAVE, the process agent controller triggers the execution of all registered outbound process agents. From this point forward, the service providers can no longer change their business object data in main memory. Instead,

process agents get read-only access to this data. This guarantees that the process agents read the same data for assembling the message as the service providers store in the database. The assembled message is passed to ESI. After the process agents are done, the SAVE method of all relevant service providers is called to store the data in the database.

5.3.4.2 Communication Patterns

In AP / ByDesign, the following communication patterns are defined in accordance with the transaction patterns introduced by the unified modeling methodology (UMM) of UN/CEFACT.[3] These communication patterns describe meaningful message choreographies between sender and receiver. The first two listed below enable transactional processing across deployment units. The others are used for data provisioning.

- Request/confirmation
  Is the basic pattern for transactional processing. It assumes that a request is sent to a process component, which, either immediately or later, sends a confirmation of the execution of the process step to the requester.
- Notification
  Describes a formal information exchange between sender and receiver. A receiving deployment unit gets a business event notification with no reaction expected by the sender other than an acknowledgement.
- Query/response
  Is the basic data provisioning pattern. Certain data is requested from and provided by the receiving process component in a read-only manner.
- Information
  Describes an informal information exchange without non-repudiation requirements. It informs predefined subscribers about data changes.

Especially in the case of the request/confirmation and query/response pattern, process agents monitor the message ex-

[3] In this book we talk of communication patterns instead of transaction patterns in order to avoid confusion with transaction interface patterns introduced in chapter 3.3.3.4.

change. They relate sent and received messages based on information stored in the process integration persistence. If the message exchange does not comply with the communication pattern, the process agent initiates appropriate actions. If, for example, a confirmation is received that refers to an outdated request, the confirmation can be ignored or a conflict can be reported.

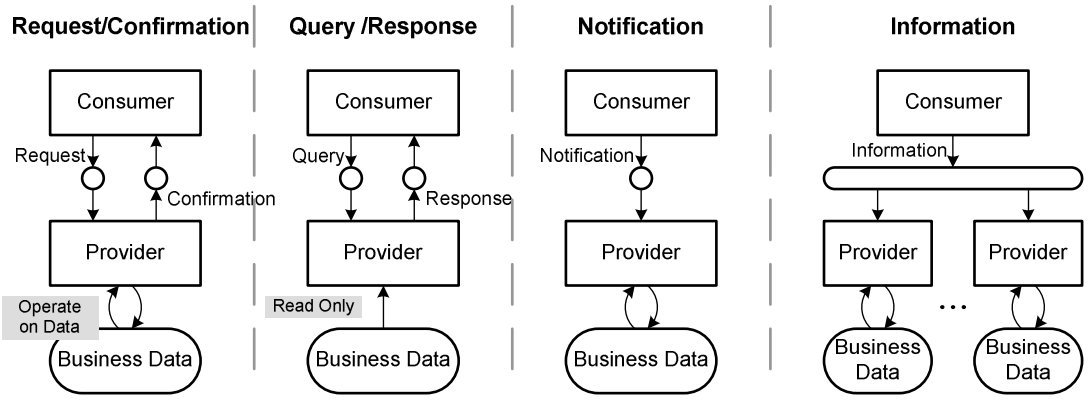


Figure 5-7 Communication Patterns

5.3.4.3 Reliable Messaging and Reconciliation

Asynchronous messaging requires a strategy to ensure that the processing of multiple messages that update the same business object instance retain a consistent state. One strategy is the usage of delta information instead of the transfer of absolute values. For example, if inventory is increased from 100 to 102, the delta would be “+2” whereas the absolute value is “102”. Delta messages eliminate dependencies between updates. The order in which messages are processed is not relevant. However, lost delta messages result immediately in data inconsistencies, which can be corrected only by reconciliation between sender and receiver (see below).

In the case of absolute value transfer, the order in which messages are processed is crucial. The following situations have to be considered carefully:

- A change message is received, but the business object to be changed does not yet exist because the create message has not arrived.
- A cancel message is received before the create message.

- A second change message is received before the first change message.

To avoid these situations, ESI offers EXACTLY ONCE AND IN ORDER processing of messages according to Web Services Reliable Messaging (WSRM). Nevertheless, lost or corrupt messages can lead to inconsistencies that prevent the processing of subsequent messages. This results in a blocked inbound queue. To solve the problem, the sequence must be restarted, which requires an initial reconciliation of sender and receiver.

Regardless of whether delta information or absolute values are transferred using asynchronous communication, it is necessary to synchronize sender and receiver from time to time to correct inconsistencies. Reconciliation establishes synchronization points between sender and receiver to maintain an overall consistent state. A reconciliation message contains an absolute value (no delta) of a business object instance and in this way resolves inconsistencies caused by lost delta messages or blocked queues.

The process agent framework provides the reconciliation mechanisms. Reconciliation can be triggered automatically, manually, or periodically.

5.4 Error and Conflict Handling in Asynchronous Communication

In addition to the consistency issues discussed in the previous section, asynchronous messaging also requires specific ways for handling error and conflict situations that prevent an inbound process agent from processing messages in a regular way:

- An error means that the process can’t continue, for example because a business object received invalid data with an incoming message (for instance error “ISO code not known”), or the relevant business object instance does not exist.
- Conflicts occur if the incoming message is based on a state that does not fit with the business object’s actual state. For example, if an incoming update of a purchase order reduces the quantity of ordered product, but the delivery of the product has already taken place.

Due to the nature of asynchronous communication, errors and conflicts have to be handled on the receiving side (forward error recovery). Simply put, the sender of the message cannot be asked directly to solve the problem. This differs markedly from the synchronous case, where errors and conflicts are reported back to the sender immediately by an error message. The sender solves the problem and the business process continues

5.4.1 Symptom, Cause, and Resolution

In order to resolve errors and conflicts, the symptom, which describes how the issue was manifested, and the cause, meaning the reason why the situation occurred, have to be evaluated. The following table shows some examples of symptoms, the possible causes, and the corresponding resolution strategies.

Examples of Symptom, Cause, and Resolution

Symptom	Possible Cause	Resolution
Configuration unknown	Invalid data or format from sender	Correct message and retry (notify sender about changed message)
	Improper environment on the receiver side (configuration or master data)	Send a notification of failure or request for correction
		Change the environment and reprocess
Late request conflict	Subsequent processing not possible on the business level	Resolve conflict on business level either automatically by sending a confirmation or manually by a user
		Investigate manually
		Reject on the business level

At configuration time, the developer or business expert defines the order of possible causes depending on the probability and preferences on the business level, along with the most likely resolution for a symptom. If possible, the resolution should run automatically; otherwise, the situation must be resolved manually.

5.4.2 Error Resolution

If the inbound process agent detects an error or conflict, it determines a symptom identifier to trigger error and conflict handling and creates an error message for the user:

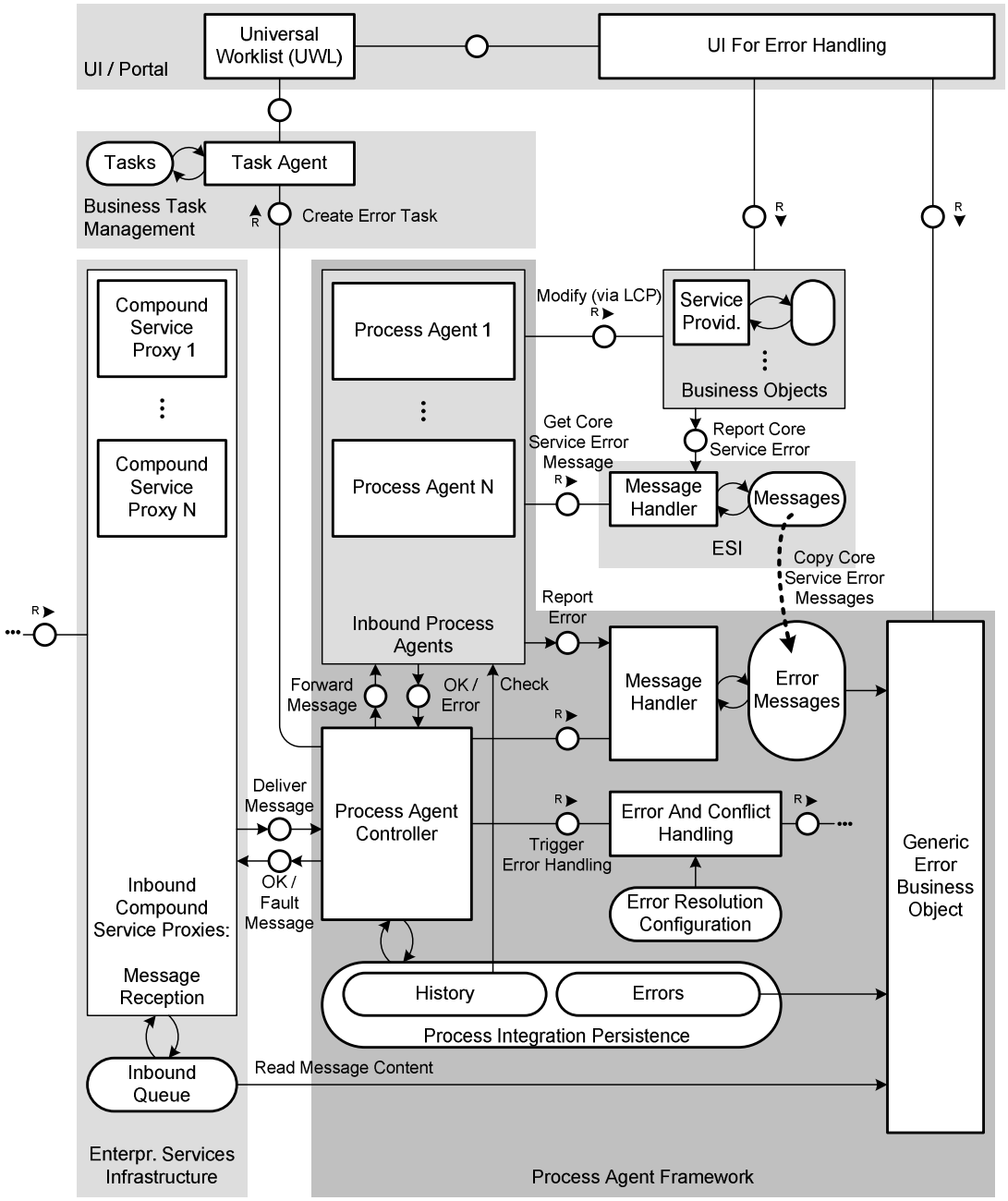


Figure 5-8 Error and Conflict Handling During Inbound Processing

- The symptom identifier is used to look up the error resolution in the configuration tables for error and conflict resolution. The error resolution method is applied.

- The error message is stored in the message handler of the process agent framework. Typically, the error message includes the names of the business objects and process components the received message refers to. If the user has to resolve the error or conflict, the corresponding error message is displayed.

In general, temporary as well as permanent errors are handled by forward error recovery. In this way, temporary errors such as locked business objects are resolved automatically by reprocessing the message after a certain time delay. Permanent errors and conflicts, such as field format errors, value mapping errors, or customizing conflicts, require user interaction, which is initiated using business task management (see below).

Additionally, a fault message is created in all error cases, informing the transport layer of ESI about the occurrence of an application error in the process agent framework. A notice is stored in process integration persistence for later tracking of errors and conflicts.

5.4.3 Error Resolution with Business Task Management

In AP / ByDesign, a task is a piece of work to be accomplished by a person. Tasks are initiated by a task agent, which determines the responsible business expert and forwards the task to the business expert's to-do list (universal work list, see chapter 6.1.1.1). Business task management enables end users or an application to create, receive, manage, and complete tasks.

If an error has to be resolved manually on the receiving side, the process agent framework creates an error task using business task management. The user finds the error task in his universal work list. If the user selects and executes an error task, the error handling user interface (UI) is displayed. It provides access to the data of the received message that caused the error and the corresponding error message. In addition, the user can display the current persisted state of the business object instances that the erroneous message refers to in a separate window.

The error handling UI provides the following functions to solve the error:

- Edit the message data and save it as a new version in the inbound queue

- Restart a single message or perform a mass restart of all messages that belong to the same error symptom, sender, and service interface
- Simulate inbound processing to display the error situation

If the problem cannot be solved by the receiver, backward error handling is initiated by sending a NOTIFICATION OF FAILURE message back to the sender. The NOTIFICATION OF FAILURE message includes the reference ID of the failed message. On the sender side it triggers an automatic reconciliation.

5.5 Synchronous Communication

In AP / ByDesign, process components normally interact asynchronously in order to achieve loose coupling. Synchronous communication is used only in specific cases when business requirements enforce a direct reply of the service provider.

5.5.1 Process Agents for Synchronous Communication

The process agent framework supports dedicated synchronous process agents for synchronous communication. The execution steps of synchronous process agents closely resemble those mentioned in section 5.3.1 and 5.3.2 for asynchronous process agents. In contrast to asynchronous process integration, a synchronous outbound process agent waits for the response of the compound service called in another deployment unit. When the call returns, the local process continues. In this way, a synchronous compound service call is used to perform an intermediated process step triggered from the local process.

When using synchronous communication, we have to distinguish between synchronous calls that only read data and synchronous calls that modify data at the receiving process component. Therefore, process agents for synchronous communication are differentiated in the following way:

- Synchronous outbound/inbound process agent for read access
- Synchronous outbound/inbound process agent for write access

As synchronous write access to other deployment units or remote components is quite challenging (see section 5.5.3), synchronous communication should be restricted to read access, if possible; for example, when checking credit card validity. By design, synchronous inbound agents for read access are not able to modify business object data; they only read data.

5.5.2 Execution of Synchronous Service Calls

Regardless of synchronous or asynchronous communication, outbound process agents are decoupled from the business logic performed in the service provider classes. Asynchronous outbound agents are registered on business object change events published by the ESI change handler (see section 5.3.1). Synchronous agents are registered on intermediate process steps, which can be called by service provider classes. A synchronous call is executed as follows:

Whenever a business object needs to perform a compound service call synchronously, it calls a predefined intermediate process step, for example PRODUCTATPCHECKREQUEST. The process agent controller receives the intermediate process step and triggers the corresponding synchronous outbound process agent to execute its steps. These steps are (see also section 5.3.1):

- Evaluate process relevance
- Condition evaluation
- Receiver determination
- Assembling of the request message

The agent controller passes the message to ESI, which establishes synchronous communication with the receiving process component where a synchronous inbound agent receives the message via the corresponding inbound compound service interface. The inbound agent processes the message and sends the result message back via the same compound service interface. On the service consumer side, the synchronous outbound agent waits until the reply message is received. The outbound agent checks the received message for correctness and transfers the reply to the service provider class that requested the intermediate process step.

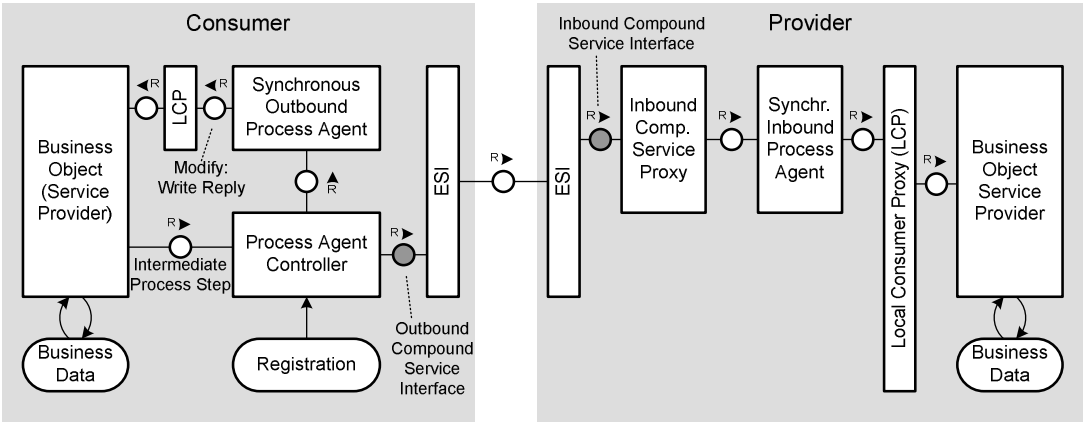


Figure 5-9 Synchronous Communication

If the receiving process component is unavailable or unable to fulfill the request within an acceptable timeframe, the call returns to the outbound process agent with a timeout, indicating that the communication partner cannot be reached or is not able to act on the request. The local service provider class continues, possibly by falling back to executing the intermediate process step asynchronously later on. This avoids the tight coupling of components typically imposed by synchronous communication.

5.5.3 Synchronous Modifications

There are situations that require synchronous compound service calls that modify business object data. One prominent example is the available-to-promise (ATP) check. The ATP check determines if a PRODUCT can be confirmed in a SALES ORDER based on whether enough stock is available in INVENTORY or can be produced or purchased in time. The PRODUCT is then reserved in the confirmed quantity.

However, direct synchronous modification across deployment units has the following challenges:

- If the LUW of the calling service provider class is rolled back for some reason, modifications of persistent data at the receiving process component must be undone also.
- If communication breaks down during a synchronous call that modifies data, the calling service provider class does not know whether the modifications have been performed at the remote component. Complex reconciliation mecha-

nisms need to be executed to restore a consistent state between sender and receiver.

To avoid this, synchronous modifications to persistent data in AP / ByDesign are made tentatively until the local LUW is committed. To make the (exceptional) synchronous write scenario reliable, a tentative update and confirm/compensate (TUCC) protocol was introduced. TUCC can be used in the typical “reservation scenario” where reservation information must be stored, but can be confirmed only at the end of the interaction.

5.6 In a Nutshell

In AP / ByDesign, process components of different deployment units are loosely coupled. They communicate via compound service interfaces using asynchronous message exchange, with service consumers (sending process components) and service providers (receiving process components) running in separate LUWs. Synchronous, message-based communication across deployment units is used only in exceptional cases.

Process agents act as mediators between business objects and compound service interfaces. Outbound process agents for asynchronous communication are registered on the event BUSINESS OBJECT CHANGED. If triggered by the process agent controller, outbound process agents execute a series of predefined steps in order to detect if a message needs to be sent and to assemble the message if required. Afterwards, ESI transfers the message via the outbound interface to the receiver. At the receiving side, the message is passed through an inbound interface to the corresponding inbound process agent. This agent checks the message and calls the core services of the targeted business objects.

The service consumer, using intermediate process steps, directly triggers outbound process agents for synchronous communication. Inbound process agents send back the response directly after the service request has been executed.

To ensure data consistency, message exchange between deployment units follows specific communication patterns. The process integration persistence is used to keep track of send and receiver messages.

If a message cannot be processed during asynchronous communication because of an error or conflict, forward error recovery

takes place using automatic resolution or human interaction triggered via business task management. For synchronous communication, error handling is done by the sending process component.

6 Constructing the User Interface

Traditional business applications were designed primarily with the expert user in mind. They centered on databases with transactions entering data and reporting tools for getting data out. Self-service applications for occasional, casual and non-expert users played a minor role. Also of less importance was automating processes that involved a lot of interaction between the end user and the application.

Unlike that traditional approach, today's software applications are designed around business processes and the people who participate in those processes. These applications have expanded into the fuzzy world of collaboration, cooperation, and decision support with new self-service applications providing much better support for business process automation and fast access to data and transactions. This allows significant improvements of people effectiveness and efficiency, which is also known as productivity.

Increasing people productivity is a key differentiator for advanced business solutions. Expert users may require complex, information-rich user interfaces (UI). However, casual users who perform a task rarely may require step-by-step instructions to get the task done. Yet both user interfaces deal with the same service-enabled business objects.

To provide users with an appropriate user interface, the following key design principles have been derived for AP / ByDesign:

- Active user interfaces
  An active user interface pushes information and tasks to users and triggers their work. It allows users to react directly on certain events.
- Readiness to work out of the box
  An intuitive screen composition and a clear navigation scheme provide the user with as much guidance as necessary to perform a task.
- Adaptable user interfaces
  A role concept allows compiling a user experience that is tailored exactly to the user's profile and job.

- Consistent user interfaces
  The UI concept addresses the unification of the user interface across all areas, providing a consistent navigation scheme as well as a consistent look and feel.
- Integration of applications, data, and technologies
  The user interface seamlessly integrates data and applications from various sources and different UI technologies. In particular, the approach of embedded analytics combines transactional and analytical applications within one user interface.
- Training on the job
  The UI concept includes knowledge transfer features that support self-directed learning tailored to user's needs.

All these design principles are guiding the UI concept of AP / ByDesign, which centers on control centers, work centers and application UIs as described in the following sections.

6.1 Organizing the End User's Work

Optimizing user productivity – that is, optimizing the effectiveness and efficiency of the user – is the key focus of the AP / ByDesign UI concept. To enable users to be effective (that is, "to do the right things"), the UI concept introduces control center and work centers (see section 6.1.1). To enable users to be efficient (that is, "to do things right"), the user interface follows a consistent design approach throughout all applications. This means that the user interface for operating on a specific aspect of a business object always looks and behaves in the same way for all business objects. To achieve this, user interfaces are designed and built from standardized UI building blocks (see section 6.1.2).

The tasks an end user has to perform depend on the user's roles in the enterprise. Each person can hold several roles such as being a manager, a sales representative, and simply an employee. Depending on these roles, the person has access to certain information and functions.

Example of Roles

Peter Greene works in the purchasing department. Therefore, he has the role PURCHASING AGENT with access to work centers called PURCHASE REQUESTS AND ORDERS, MANAGING PURCHASING,

SOURCING AND CONTRACTING, and VENDOR INVOICING (see figure 6-2). Of course, he is an employee of the enterprise, so he also has the role EMPLOYEE, which gives him access to company news and to some employee self-services such as LEAVE REQUEST.

In general, a user has the authorizations for all business objects and functions of the work centers assigned to him. Authorization checks are performed to ensure that users initiate only user interfaces assigned to their roles and work centers. To ensure this, existing mechanisms in the portal content directory (PCD) and the user management engine (UME) are used. In the back end, authorizations are checked using identity and access management (see chapter 4.3.4).

6.1.1 User Interface Structure

In order to optimally organize the end user's work, the AP / ByDesign UI is structured in control centers, work centers, and application UIs:

- The control center is the home area of a user and provides him with the most important information about his daily work. Whenever a user logs on to AP / ByDesign, he starts his control center.
- A work center is a user's business access and control area for a specific work area, such as PURCHASE ORDER MANAGEMENT or SALES ORDER MANAGEMENT.
- For specific tasks such as "Create sales order", the user navigates from the work center to the corresponding application UI.

Which work centers and application UIs are offered to a user is controlled by his roles. The user's control center aggregates all work centers and selected application UIs assigned to his roles, as shown in figure 6-1. The access path from control center via work center to application UI provides the user with clear navigation.

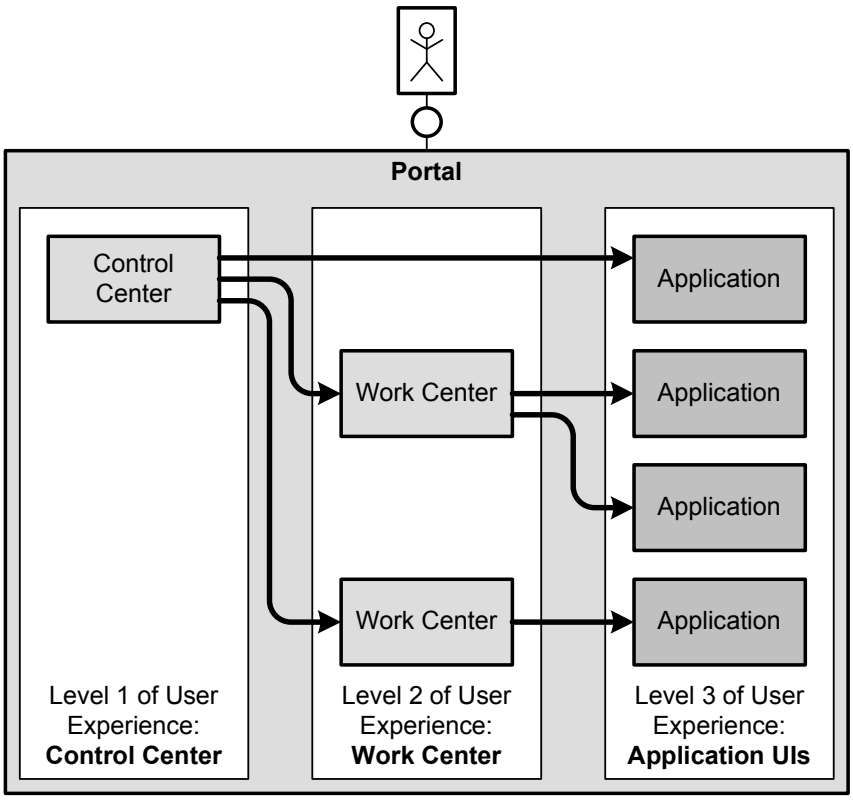


Figure 6-1 Navigation Between Control Center, Work Centers, and Application UIs

6.1.1.1 The User's Home Area: Control Center

A control center aggregates, summarizes, and provides all information the user needs for organizing, monitoring, and executing his daily work (see figure 6-2). This includes news, reports, and an overview of the user's work items. In addition, the user's universal work list (UWL) is part of the control center that pushes work items, such as tasks and alerts, to the user and actively triggers his work. This push mechanism is part of the AP / ByDesign active user interface approach.

Figure 6-2 Control Center

Figure 6-2 highlights the following two navigation options the user has with the AP / ByDesign UI: top-level navigation and contextual navigation.

- The top-level navigation provides navigation between the control center and the work centers. The example shows four work centers assigned to Peter Greene: PURCHASE REQUESTS AND ORDERS, MANAGING PURCHASING, SOURCING AND CONTRACTING, and VENDOR INVOICING. The control center is labeled as MY HOME.

- The contextual navigation panel on the left offers the user access to different views of the page as well as links to operations or business object instances. For instance, the control center organizes a user's daily work by control center views like work, news, and reports. In addition, the user can CREATE TASK or navigate to PURCHASE ORDER 214559. Information accessed via contextual navigation is always displayed in the central content area.

Object-based navigation provides a third navigation option, and is described in section 6.2.4.1.

6.1.1.2 The User's Work Area: Work Center

A work center organizes and supports all user activities in a specific business area, for example purchase order management. The information and data needed to perform certain activities are categorized and presented as work center views. Figure 6-3 shows the work center PURCHASE REQUESTS AND ORDERS, which consists of four work center views: OVERVIEW, PURCHASE REQUESTS, PURCHASE ORDERS, and REPORTS. As with control centers, the contextual navigation panel on the left provides navigation between those views.

The work center provides access to the application UIs a user needs for his work via the contextual navigation panel, as we show here, or via object-based navigation (see section 6.2.4.1).

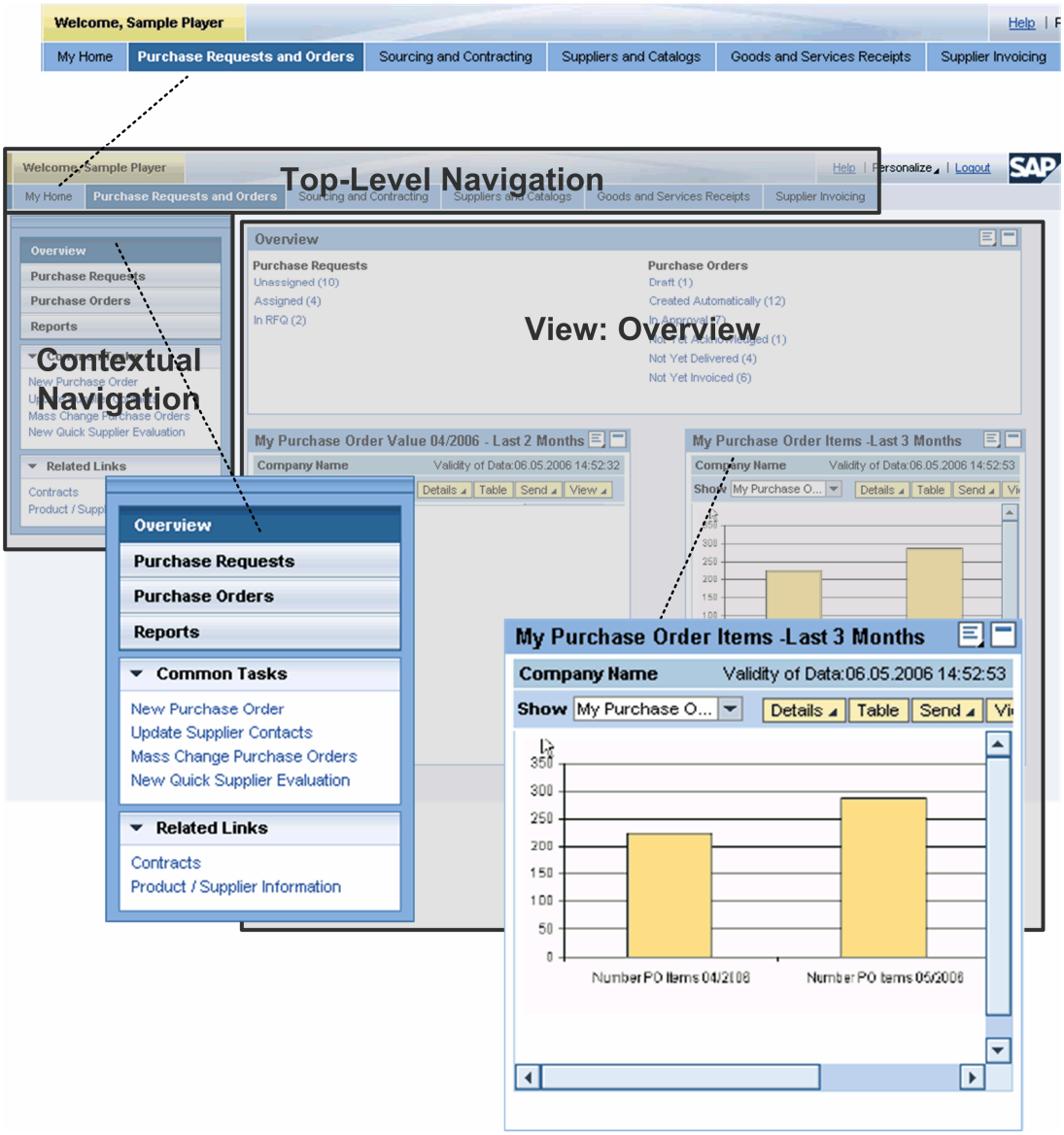


Figure 6-3 Work Center Purchase Requests and Orders Overview

6.1.1.3 The User's Application Area: Application UIs

Application UIs enable users to fulfill specific tasks. Figure 6-4 shows an application UI that allows the user to maintain purchase order items of purchase order 54954843. Application UIs follow specific UI and interaction patterns. A floor plan defines the basic structure of an application UI (see next section).

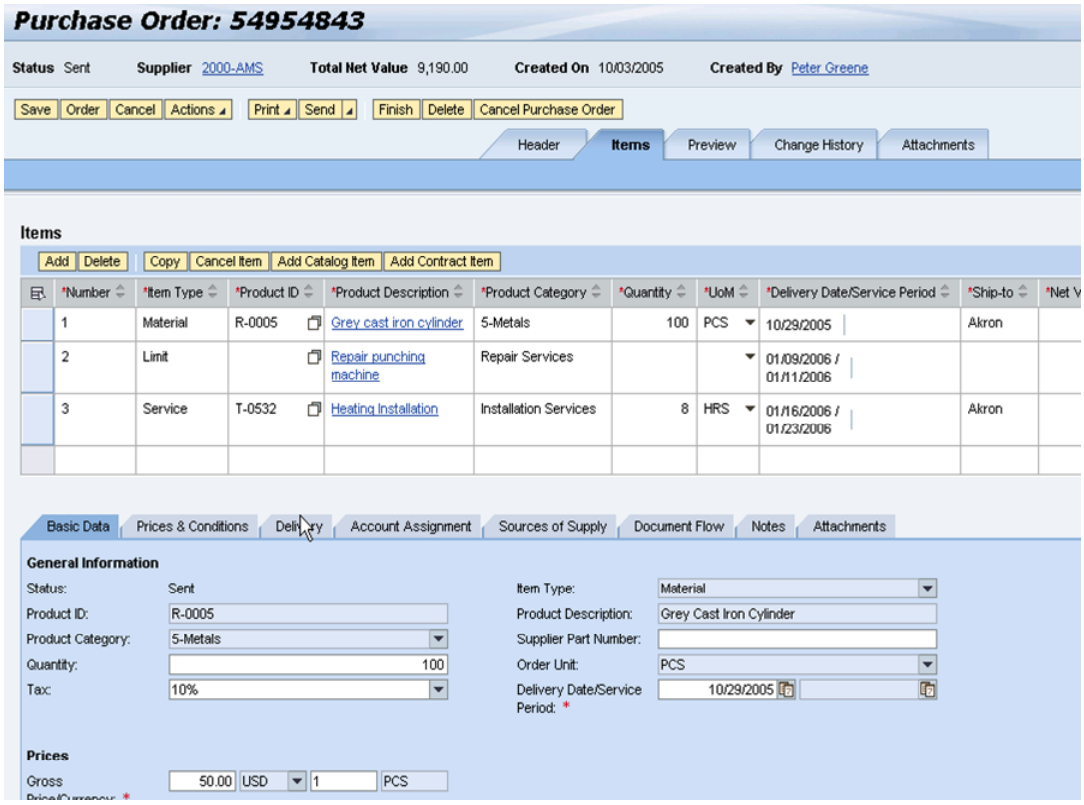


Figure 6-4 Application Area: Maintain Purchase Order

6.1.2 User Interface Building Blocks

With regard to designing user interfaces, we make a fundamental distinction between graphic design and interactive design.

- *Graphic design* is concerned with the interface layout. Standardized format templates (style sheets) determine which colors, fonts, and sizes are used in the interface. Usually the graphic design of a software application is modified to match the layout an enterprise already uses. Layout modifications to a style sheet are done by using a style sheet editor that is delivered, for example, with the SAP NetWeaver® Portal.
- *Interaction design*, on the other hand, is not concerned with colors and fonts, but with how the application itself can support users in their single work steps and workflow, and how to structure the user interface accordingly. Interaction design determines the structure of the user interface: which data can be displayed where and when, which interface elements are used for entering data, and how navigation to functions, applications, and information works.

The AP / ByDesign interaction design is based on standardized, reusable user interface building blocks (UI building blocks). We differentiate between three types of UI building blocks:

- Floor plan
- UI pattern
- UI element

A floor plan defines the basic structure of an application UI. A floor plan represents a generic user work scenario and is designed in accordance with the underlying interaction requirements. It combines multiple UI patterns in a certain order.

A UI pattern is a UI building block that serves one dedicated user task; for example, searching for and identifying business object instances, inspecting and maintaining attributes, or analyzing data. In general, a pattern describes a common, well-known solution to a recurring problem. Repetitive use of patterns by user interfaces increases consistency and makes the user interface more intuitive and easier to use.

UI patterns can be applied in different floor plans and also within other UI patterns. One UI pattern is built from multiple UI elements. A UI element defines an elementary part of an UI pattern. Typical UI elements are buttons, tables, trees, fields, and labels.

In the following section, we introduce three examples of floor plans.

6.1.2.1 Object Instance Floor Plan

Object instance floor plans allow end users to display and modify all attributes of a given business object instance.

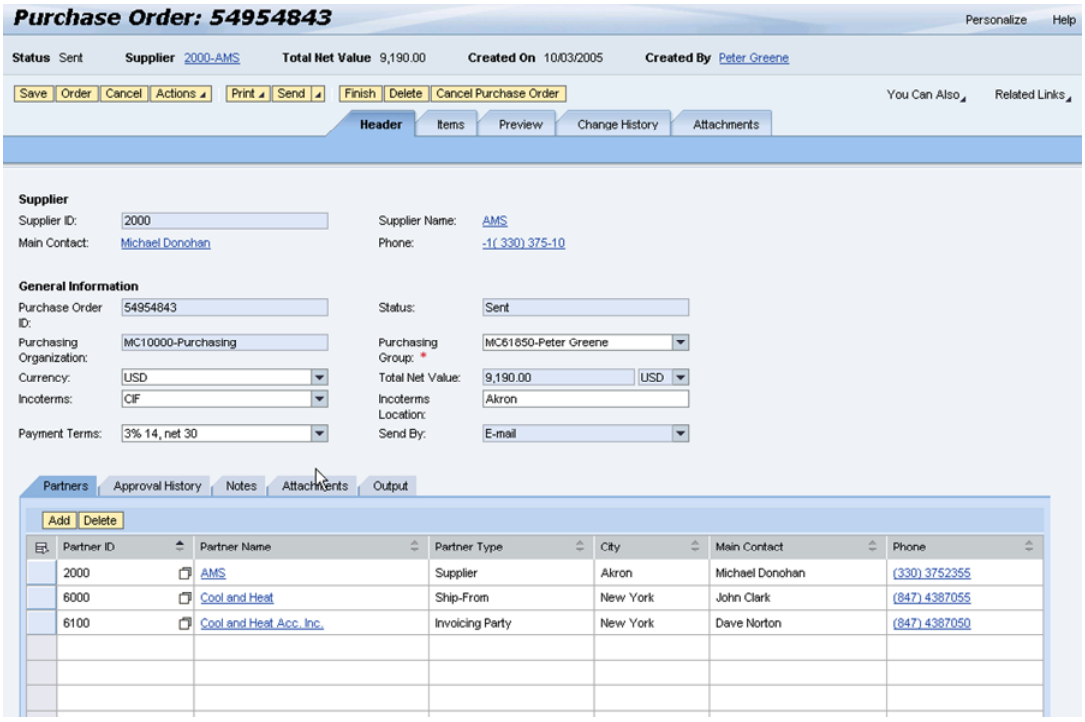


Figure 6-5 Object Instance Floor Plan

The object instance floor plan consists of a navigation area on the upper right and the content area (see figure 6-5). The navigation area consists of the UI pattern YOU CAN ALSO, and RELATED LINKS. In the content area, each tab is represented by one UI pattern, such as the form pattern or the list pattern.

6.1.2.2 Quick Activity Floor Plan

In contrast to the object instance floor plan, which enables handling of all attributes of an given business object instance, quick activity floor plans are used to display and modify only the most basic attributes. This floor plan embeds the following UI patterns (see figure 6-6):

- The title with text and message area
- A form pattern
- A list pattern together with the core functionality toolbar

A comparison with the object instance floor plan shows that UI patterns are reused within different floor plans: in this case the form pattern.

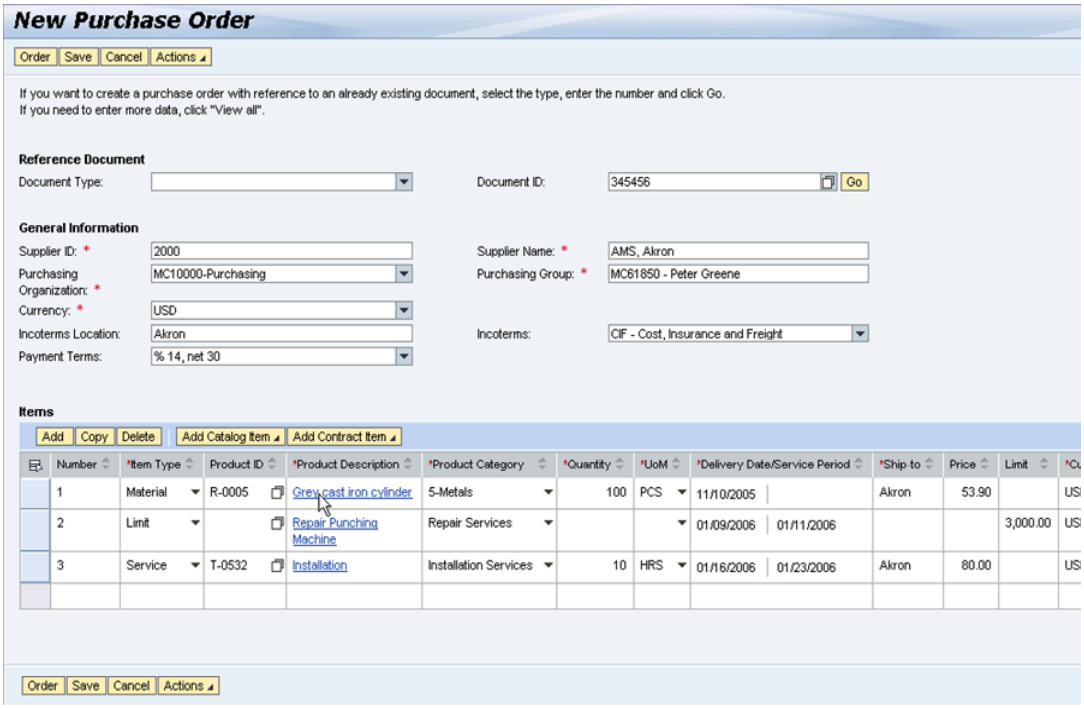


Figure 6-6 Quick Activity Floor Plan

6.1.2.3 Guided Activity Floor Plan

The guided activity floor plan guides the end user step by step through an activity (see figure 6-7). In general, multistep activities can be accomplished more easily with screens based on the guided activity floor plan than with screens based on the object instance floor plan.

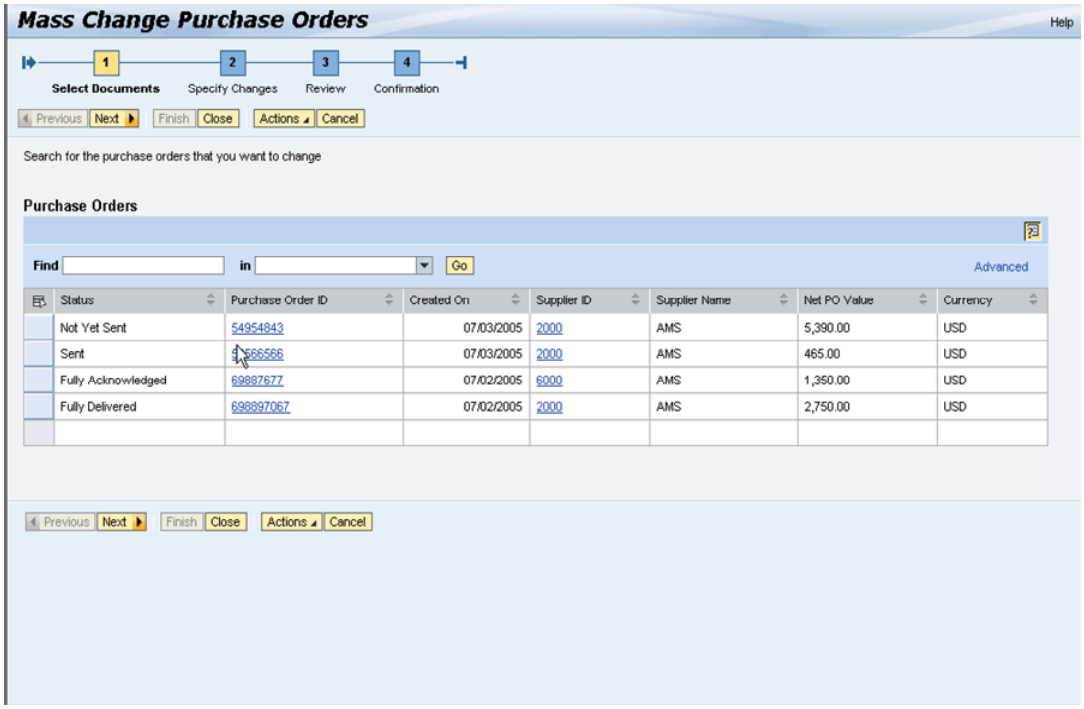


Figure 6-7 Guided Activity Floor Plan

6.1.3 Embedded Analytics

Although there are use cases where analytical UIs incorporate an entire screen, for example when presenting a planning application, usually reporting and data analysis need to be integrated in transactional user interfaces and processes. For that purpose, there are two analytical UI patterns that can be embedded in a floor plan, a work center, or a control center:

- Information consumer pattern for presenting simple tables, pivot tables, and graphics with limited functionality
- Analysis pattern supporting advanced data analysis with focus on end-user interaction

There is the additional requirement for displaying reports according to a dedicated format and layout defined by legal regulations or customer-defined conventions. This type of reporting cannot be handled by a UI pattern because the look and feel of such formatted reporting differs from report to report.

6.1.3.1 Information Consumer Pattern

The information consumer pattern can be embedded into control centers, work centers, and floor plans. It shows the most important

figures and business object instances for a specific user. When integrated into a floor plan, the information consumer pattern typically displays a partial view of data sufficient to complete a task. Services such as exporting data to Microsoft Excel, sending it by e-mail, or printing it are available within the information consumer pattern. However, in comparison with the analysis pattern, the end user has very limited interaction possibilities.

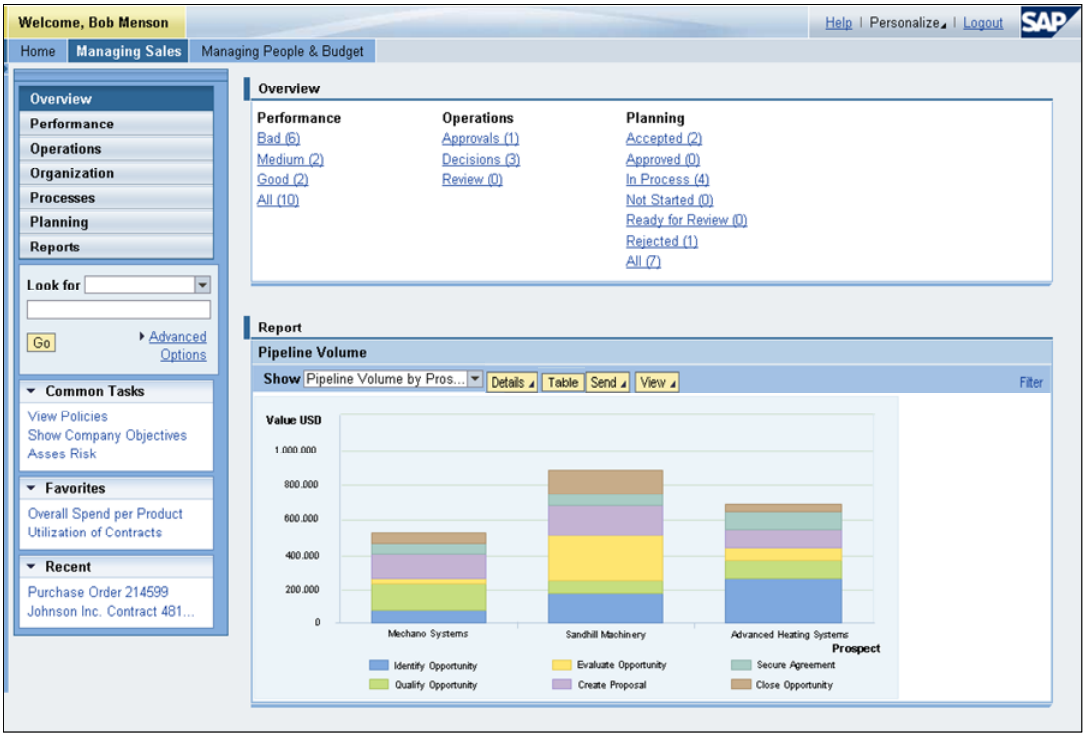


Figure 6-8 Example of Information Consumer Pattern

6.1.3.2 Analysis Pattern

The analysis pattern offers the user all functions relevant for analytical reporting. The analysis pattern can be used as both a detailed view of an information consumer pattern or directly embedded in control centers, work centers, or other floor plans.

The analysis pattern provides two fundamental capabilities: detailed analysis and view definition. Within the analysis pattern, the user can define and persist personal views on the data.

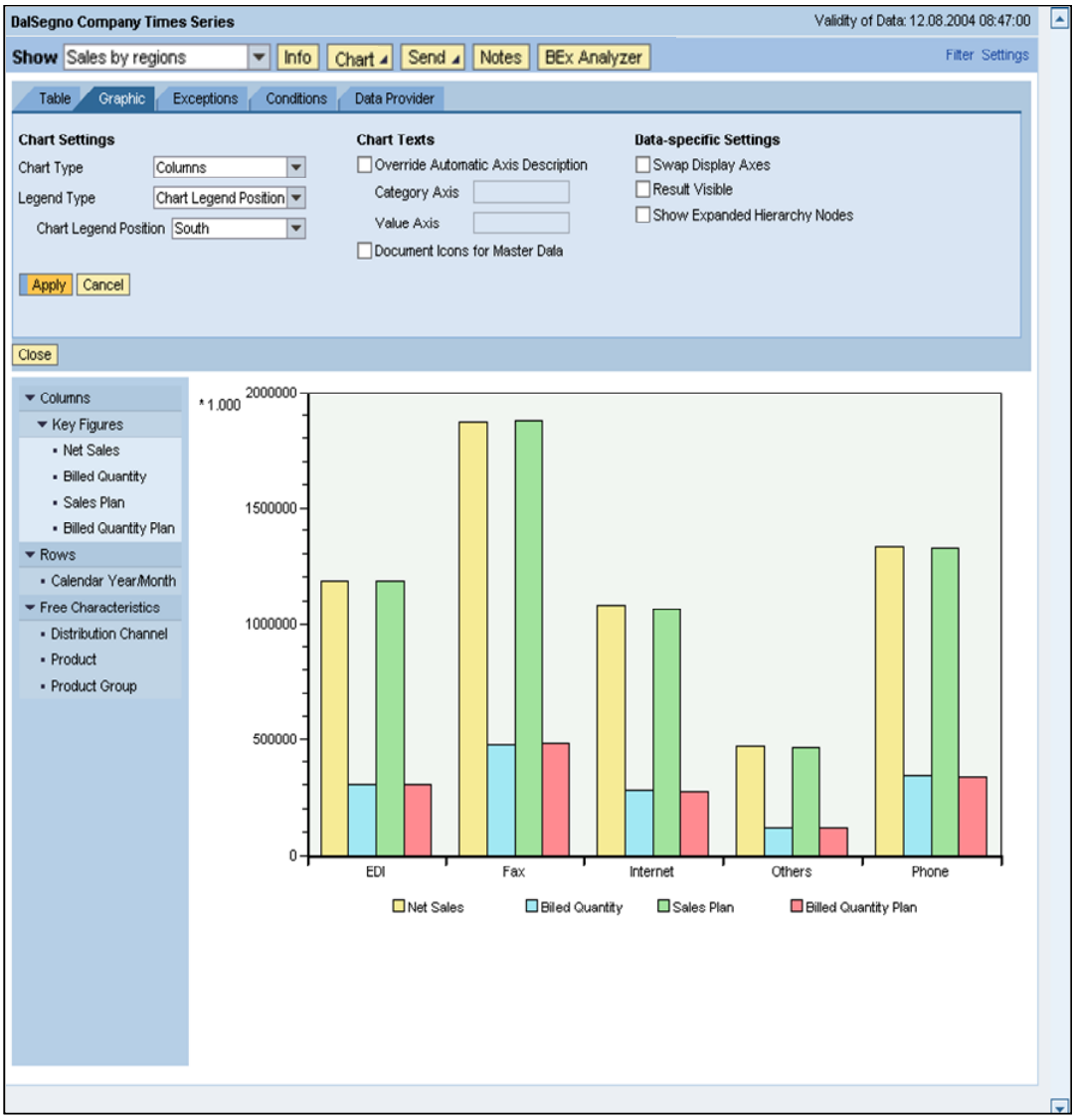


Figure 6-9 Example of Analysis Pattern

6.1.4 Built-In Learning Environment

Users should be able to operate business applications with confidence and minimal time investment in training. Therefore, AP / ByDesign integrates access to all relevant learning content right from the user interface. Animations show the user how to work with the user interface thereby reducing the amount of documentation that the user needs to read. Simulations let the user interact with the application UI without modifying actual business data.

The user can access built-in learning environment using two knowledge providers:

- Learning center

The learning center is part of the control center and provides guided knowledge using self-service learning maps. A personal knowledge scorecard helps the user track his learning progress and identify remaining open topics. The content is adaptive to the specific user's needs (by language, industry, role, and so on) and to business configuration to present only "just-in-context" information and to avoid information overload.

- Help center
  The user can reach the help center via the HELP link within the top-level navigation. All required content to understand and to work within the current work center is available, including context-specific FAQs, simulations, learning modules, and company-specific documents. The provided help content can be adapted to specific user's requirements. The user can add personalized notes to help items with the MY NOTES editor; these notes are visible only to the user.
  If the user needs additional help, he can request direct support via the help center. This concept of support is also referred to as automated service and support (see chapter 9.1).

Context-sensitive knowledge transfer together with pattern-based user interface help users become familiar with the applications in a short amount of time.

6.2 Understanding the Technology of the AP / ByDesign User Interface

The technical foundation for building and running the AP / ByDesign UI is provided by SAP NetWeaver Portal, Web Dynpro for Java, and SAP NetWeaver Visual Composer.

SAP NetWeaver Portal provides the overall infrastructure for accessing the user interface. Web Dynpro is the principal design and runtime environment for UI building blocks, while SAP NetWeaver Visual Composer provides the technology to model the user interface from standardized UI building blocks without writing code.

As described in section 6.1.2, standardized UI building blocks are a key concept of the AP / ByDesign UI. From the end-user perspective, these UI building blocks are the basis for a high level of UI consistency. From the developer perspective, they allow for

standardized interfaces between user interface and business objects. With this standardized interface approach, the user interface accesses business logic only via core services of the business objects. This leads to a strict decoupling of UI logic from business logic and makes it possible to construct any combination between UI building blocks and business objects. As a result, business object developers do not have to be concerned with the specifics of the user interface. UI designers can create multiple user interfaces for the same business object to support the needs of different user groups.

6.2.1 Portals

In general, a portal is a framework for integrating and visualizing information and applications from various sources within or across organizational boundaries. The portal runs portal applications, which are specialized Web applications[4] supporting value-added services such as personalization, single sign-on, content aggregation from various sources, and so on. A portal application consists of one or several portal components (also known as portlets), which combine program code for providing or retrieving data with appropriate user interface information. Portal components are deployed in a container that provides an appropriate runtime environment. Portal components create HTML output and communicate with the Web browser via HTTP.

6.2.2 SAP NetWeaver Portal

The SAP environment for running and administering portal applications is SAP NetWeaver Portal. It allows to access arbitrary applications and data, independent of implementation technologies and data sources, as long as common standards such as XML, HTML, or SOAP are supported. Various applications, data, and technologies can be integrated on the user interface side, providing a unified layout and consistent look and feel. Furthermore, the SAP NetWeaver Portal concept of roles and the assignment of roles to individual users allow controlling the subset of portal applications which a user needs to perform his daily work.

[4] A Web application is an application that can be accessed by a Web browser via a URL (Uniform Resource Locator).

In SAP NetWeaver Portal, each portal component together with its container is called an iView. iViews are addressable via their iView URLs and provide a set of parameterizable properties that determine, for instance, display details. iViews can be combined to define portal pages and worksets (see figure 6-10), which in turn can be assigned to certain roles.

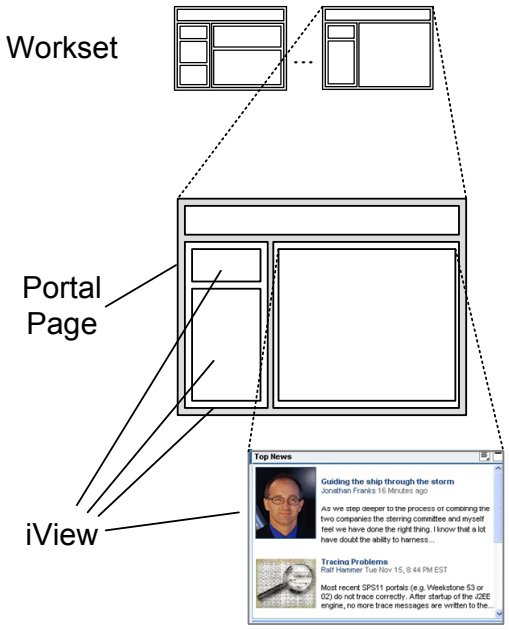


Figure 6-10 Worksets, Portal Pages, and iViews

- Portal page
  A portal page determines the physical arrangement of selected iViews in a specific screen layout. Portal pages can be assigned to a workset.
- Workset
  A workset groups various portal pages and iViews necessary to perform the tasks of a specific business activity. For instance, all iViews belonging to purchasing can constitute a workset. Worksets are usually assigned to a role.
- Role
  A role specifies the worksets, portal pages, and iViews necessary to fulfill a certain job or position in an enterprise. A navigation structure is defined for each role, which directly determines the (allowed) top level and detailed navigation within the portal.

The properties and relationships of roles, worksets, portal pages, and iViews are defined in the portal content directory (PCD). At

runtime, these definitions are interpreted for the creation of portal pages.

SAP NetWeaver Portal technology, functionality, and concepts are an integral part of the AP / ByDesign UI concept. Control centers and work centers are implemented as worksets. The control center is created at runtime from the user's role definitions in PCD.

The principal AP / ByDesign technology to build portal applications is Web Dynpro for Java. However, some features of analytical portal applications use additional technologies.

6.2.3 Web Dynpro

Web Dynpro provides an infrastructure for designing and implementing either arbitrary user interfaces (Web Dynpro Foundation) or standardized user interfaces from reusable building blocks (Web Dynpro Pattern). Reusable UI building blocks, together with the modeling tool SAP NetWeaver Visual Composer, make it possible to compose a user interface from UI building blocks rather than handcrafting it with software code.

6.2.3.1 Web Dynpro Components as Reusable UI Building Blocks

Web Dynpro follows a software architecture well known as model-view-controller (MVC) pattern, which separates the data model (model), the visible screen output (view), and the control logic to process user input (controller). Each Web Dynpro component refers to a model and consists of one or more views and controllers with the following tasks (see figure 6-11):

- Models define the application data to be presented on the user interface. In AP / ByDesign, the model acts as a representative to the business object in the back end.
- Views are used to visualize application data.
- Controllers handle communication between model and views, react to UI events, and update application data.

The model-view-controller approach ensures that the presentation logic is strictly separated from business logic and layout. Within a Web Dynpro component, a dedicated component controller handles the communication between the different view-controller pairs (see figure 6-11) and the model interface.

The user interface for a specific use case is represented by the corresponding Web Dynpro application. Basically, a Web Dynpro application is addressable by a URL, and serves as an entry point to the Web Dynpro components of the user interface. Web Dynpro applications can be integrated directly as iViews in the portal and are executed by Web Dynpro runtime (see also chapter 4.3.2).

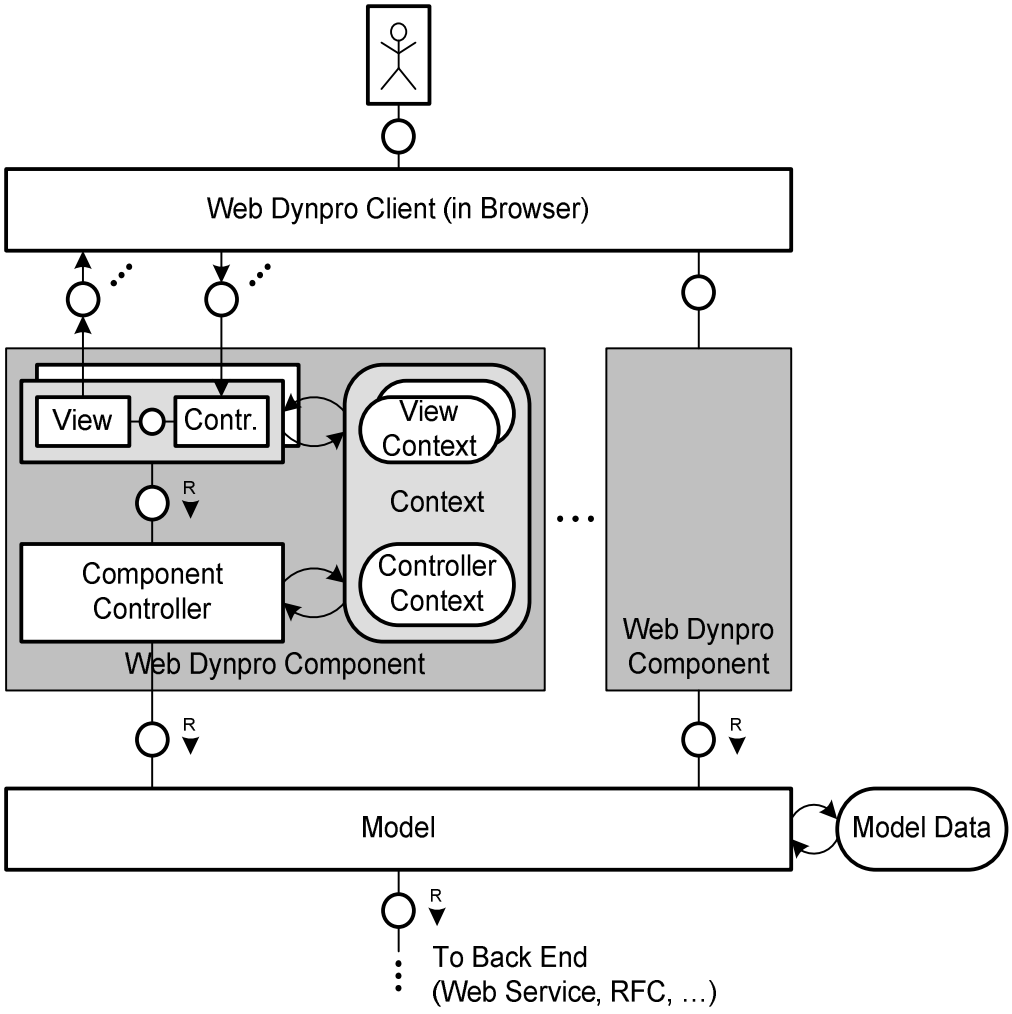


Figure 6-11 Components of a Web Dynpro User Interface

Web Dynpro components implement the reusable UI building blocks (floor plans, UI patterns, and UI elements) mentioned above (see section 6.1.2). Whenever these Web Dynpro components are used, they provide the same set of basic functions. For example, the UI element for displaying lists is implemented by the Web Dynpro component APPLICATION LIST VIEWER, which includes a set of simple, list-oriented functions, such as the following:

- Sorting
- Filtering

- Totaling and subtotaling
- Hiding columns

In AP / ByDesign, all application UIs are visualized using floor plans as the starting point; each floor plan is implemented as a Web Dynpro component. At runtime, a tree-like hierarchy of subordinated Web Dynpro components is created for each floor plan. The component controller of the floor plan interacts with the subcomponents that provide views and controllers for UI patterns and UI elements.

At design time, UI building blocks are configured and composed by the design tool SAP NetWeaver Visual Composer (see section 6.3). They are bound to the business objects, nodes, attributes, and core services provided by the back end. For example, the floor plan is bound to exactly one business object. Individual fields on the screen are bound to attributes of business objects. Buttons are bound to ACTION core services.

To enable this, a pattern kit extends SAP NetWeaver Visual Composer to model user interfaces from UI building blocks (see section 6.3.1). The UI model composed with SAP NetWeaver Visual Composer serves as configuration of the Web Dynpro components.

6.2.3.2 SAP NetWeaver Portal and Web Dynpro

For Web Dynpro applications, SAP NetWeaver Portal includes a Web Dynpro page builder that creates portal pages consisting solely of Web Dynpro iViews. The Web Dynpro page builder runs within the Web Dynpro runtime on the front-end server. Portal runtime and Web Dynpro runtime have access to the portal content directory (PCD), which contains role and user information, layout definitions, and personalization settings (see figure 6-12). The layout of portal pages is preset by the definitions in PCD, whereas the internal layout and the content of Web Dynpro iViews are provided by Web Dynpro runtime.

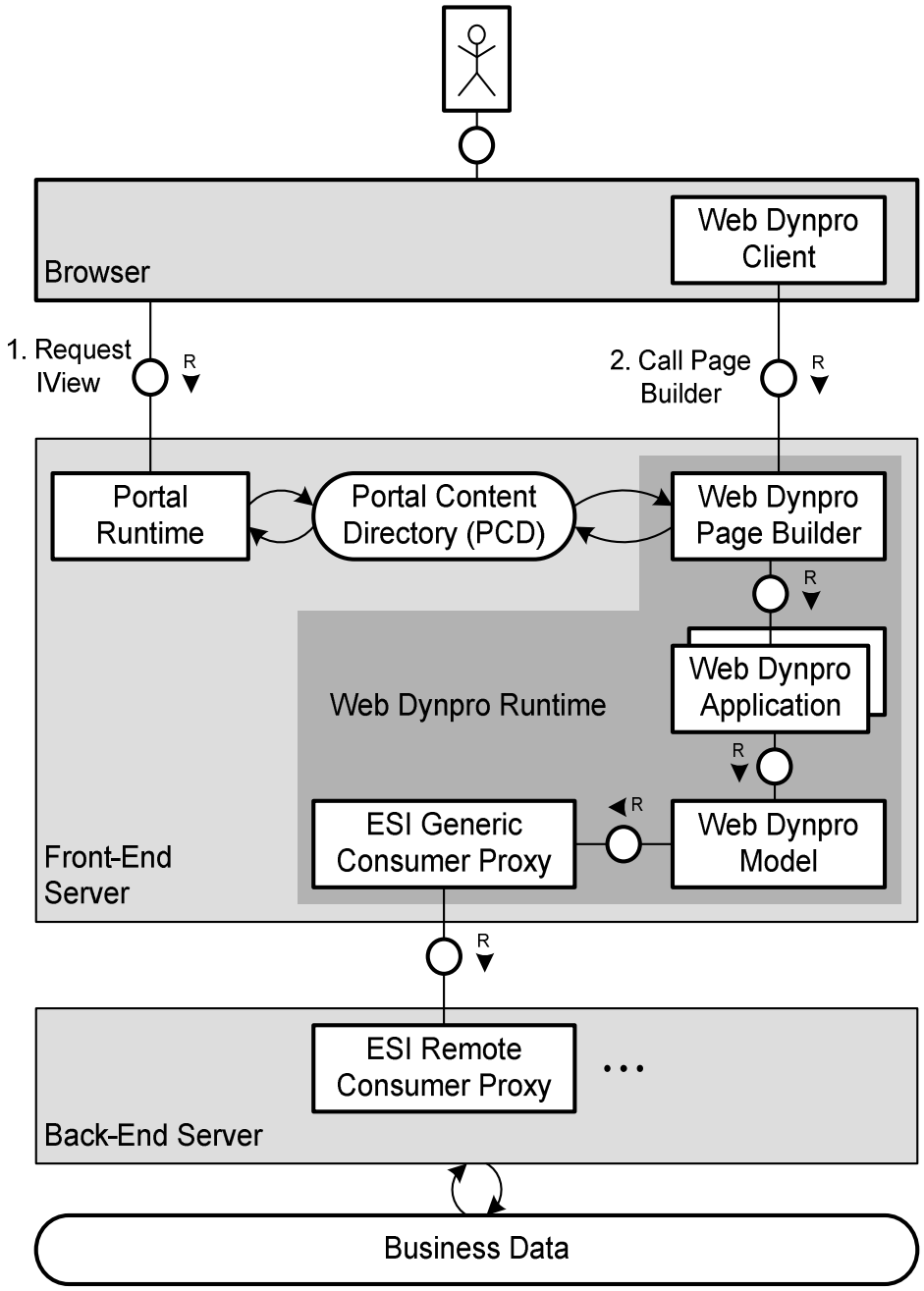


Figure 6-12 Running Web Dynpro Applications in SAP NetWeaver Portal

6.2.4 Runtime Architecture

In AP / ByDesign, the MVC pattern is implemented with one single generic model (Web Dynpro model) that handles all accesses to back-end data.

Depending on the portal user's role, an AP / ByDesign portal page offers navigation links to selected Web Dynpro applications. The following steps are executed when a portal user selects a naviga-

tion link and requests a certain application UI, for example an object instance floor plan:

1. The Web browser sends the URL of the requested iView to SAP NetWeaver Portal runtime on the front-end server (see figure 6-12).
2. The iView URL is used to look up the URL of the corresponding Web Dynpro application, which is used to call the Web Dynpro page builder.
3. The Web Dynpro page builder now starts the Web Dynpro components that are associated with the Web Dynpro application.
4. The corresponding Web Dynpro components interact with the generic Web Dynpro model for accessing the application platform via core services.
5. The Web Dynpro model represents all business objects and communicates directly with the generic consumer proxy (GCP) provided by ESI to call the business object's core services (see chapter 4.3.2).
6. All Web Dynpro components that were triggered by the request of the Web Dynpro application render their part of the user interface.
7. The Web Dynpro component that implements the floor plan forwards the rendered Web Dynpro UI together with the retrieved data to the Web Dynpro page builder.
8. The complete portal page, including navigation, is created using layout, role, and user configuration from PCD.
9. Finally, the page is forwarded to the Web browser and displayed there by the Web Dynpro client.

After all the steps are performed, the requested object-instance floor plan is displayed, and the user can proceed with his current task.

6.2.4.1 Object-Based Navigation

In addition to the standard navigation between a user's control center, work centers, and portal applications as described in section 6.1.1.1, the AP / ByDesign UI offers object-based navigation using runtime information. With standard portal navigation the target iView is identified by a URL defined at design time, while object-based navigation determines the target iView at runtime from

a set of parameters, such as the user's role, the business object type, the operation to be performed, or the business object instance ID.

Object-based navigation is also used to offer context-dependent navigation links such as in the YOU CAN ALSO pattern.

6.3 Modeling the User Interface

The user interface of AP / ByDesign is constructed from UI building blocks (floor plans, UI pattern, and UI elements). At design time, the concrete user interfaces are modeled and configured from these UI building blocks using the SAP graphical design tool, SAP NetWeaver Visual Composer.

6.3.1 SAP NetWeaver Visual Composer

SAP NetWeaver Visual Composer is a graphical modeling framework and is used for user interface design. The framework is implemented by the SAP NetWeaver Visual Composer core, which provides, among other components, a graphical user interface for modeling (called storyboard, see chapter 10.3.3), model management services, and connections to back-end systems. Extensions, known as kits, allow SAP NetWeaver Visual Composer to operate on specific model types. The pattern kit, for example, is comprised of all tools that are needed to operate on UI building blocks. Each UI building block contributes its specific tools to this pattern kit (see figure 6-13).

SAP also provides an ESI kit that enables SAP NetWeaver Visual Composer to read business object definitions, as defined in ES Repository, from back-end systems (see figure 6-13). These business object definitions can be used to model the interactions of the user interface with core service operations.

During the design phase, the following are described within a SAP NetWeaver Visual Composer model:

- Data flow between UI building blocks
- Binding of UI building blocks to business objects and core services
- Interaction between UI building blocks
- Details of the layout, such as the visibility of data

In addition to its specific tools for the pattern kit, each UI building block provides a runtime component for executing the UI building block, that is, the corresponding Web Dynpro component (see figure 6-13).

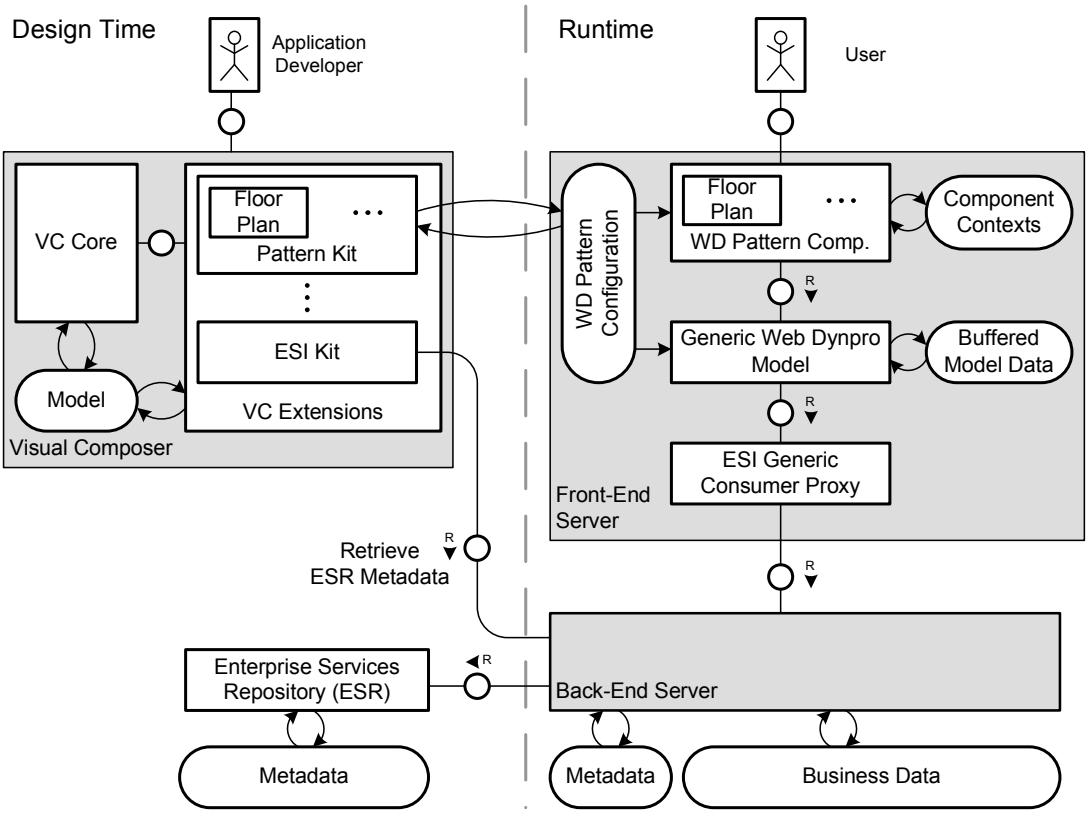


Figure 6-13 Pattern Design Time and Runtime Access to ESI

To create a new portal application, the developer starts with the selection of a floor plan. To bind the floor plan to a business object, the business object model from ES Repository is imported. Depending on the floor plan chosen and the bound business object, an SAP NetWeaver Visual Composer model is generated. The main task for the application developer now is to configure the model's entities. At the end, a concrete portal application is generated from the SAP NetWeaver Visual Composer model. When deployed to the runtime system, the SAP NetWeaver Visual Composer model is transformed to corresponding pattern configurations that invoke the respective business object core service operations (see also chapter 10.3.3).

6.3.2 Adjusting Business Object Data for Visualization

A UI design approach based on business object models and UI patterns prohibits freestyle design and enables the developer to act only within the limitation of the models and the patterns. Using its set of attributes and the data types assigned to these attributes, each business object (model) determines what can be displayed (set of attributes) and how it can be displayed (assigned data types) in a floor plan. Each floor plan defines the set of allowed UI patterns and determines the coarse-grained arrangement of those UI patterns. Therefore, both the business object models and the UI patterns influence the final presentation of data.

Furthermore, a floor plan can only be bound to one “leading” business object. Via a floor plan, an end user can only create, change, or delete instances of a floor plans leading business object. Data that is associated to the leading business object via cross-business object associations (see chapter 3.2.1.2.2) is visible but not changeable via the floor plan.

This UI design approach is sufficient for many cases and keeps the user interface as simple as possible for both the end user as well as the application developer. However, in the following cases, there is the need to go beyond the limits of this approach:

- Display data in another format as defined by the given data type
- Display data derived from business object data through calculation
- Modify data of more than one business object at the same time via a single floor plan
- Control the message flow to the user interface
- Control the screen sequence in a floor plan (for example, check whether the transition from one step to the next step makes sense in the specific business case)

The AP / ByDesign approach for adjusting business object data for visualization leaves the UI building blocks and the generic access to business objects via core services untouched. Instead, this adjustment is done either in the front end using SAP NetWeaver Visual Composer service adaptation, or in the back end by creating specific business objects or nodes tailored for the desired user interface. In the back end three new business object types are introduced to transform and combine business object data for

visualization: transformation nodes, transformed objects, and controller objects. Transformation nodes complement existing business objects, whereas transformed objects and controller objects are assigned to floor plans instead of a standard business object, thereby providing a tailored view or interaction at the user interface. In the front end, SAP NetWeaver Visual Composer provides service adaptation functionality to include data from associated business object nodes.

6.3.2.1 Transformation Node

A transformation node can be added to a business object to derive and combine data from other business object nodes of the same business object at runtime and display it on the user interface. An example for the use of a transformation node is displaying a task's processing time, which is calculated from the difference of creation time and actual date. To ensure data consistency, transformation nodes have no own persistency on the database, but access other business object nodes for retrieving and updating data. A transformation node is bound to a floor plan via a business object (see figure 6-14).

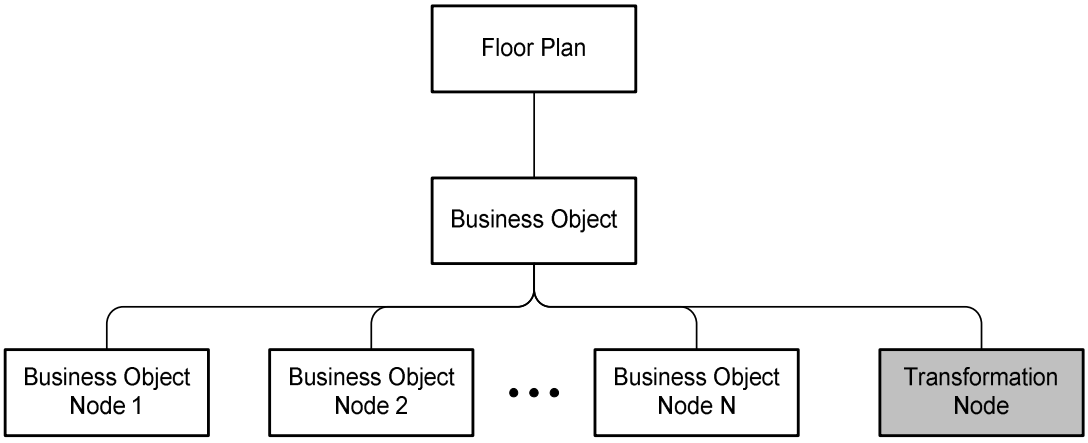


Figure 6-14 Binding a Floor Plan to a Business Object with a Transformation Node

6.3.2.2 Transformed Object

A transformed object is a specific business object type that combines attributes from multiple business objects for display on the use interface. An example is business object PARTY, which combines the separate business objects EMPLOYEE and

ORGANIZATIONAL UNIT. A transformed object can also add additional business logic to data. Like standard business objects, transformed objects consist of business object nodes and provide core services. All nodes except the root node are included from other business objects via cross-business object associations. As a result, transformed objects have no own persistence. They retrieve and save their data via the core services of the associated business objects. Transformed objects can be bound directly to a floor plan (see figure 6-15).

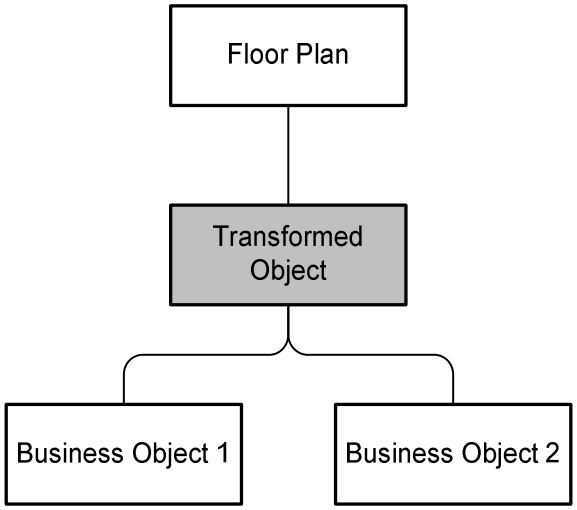


Figure 6-15 Binding a Floor Plan to a Transformed Object

6.3.2.3 Controller Object

A controller object is a technical object designed in ES Repository to change default properties and behavior of a floor plan. Unlike transformed objects and transformation nodes, a controller object is not designed for reuse but for one specific user interface. Controller objects act as mediators between floor plan and business object. For example, they control the message flow to the user interface – avoiding a message overflow of the end user – and the screen sequence for a guided activity based on a specific business situation (see section 6.1.2.3).

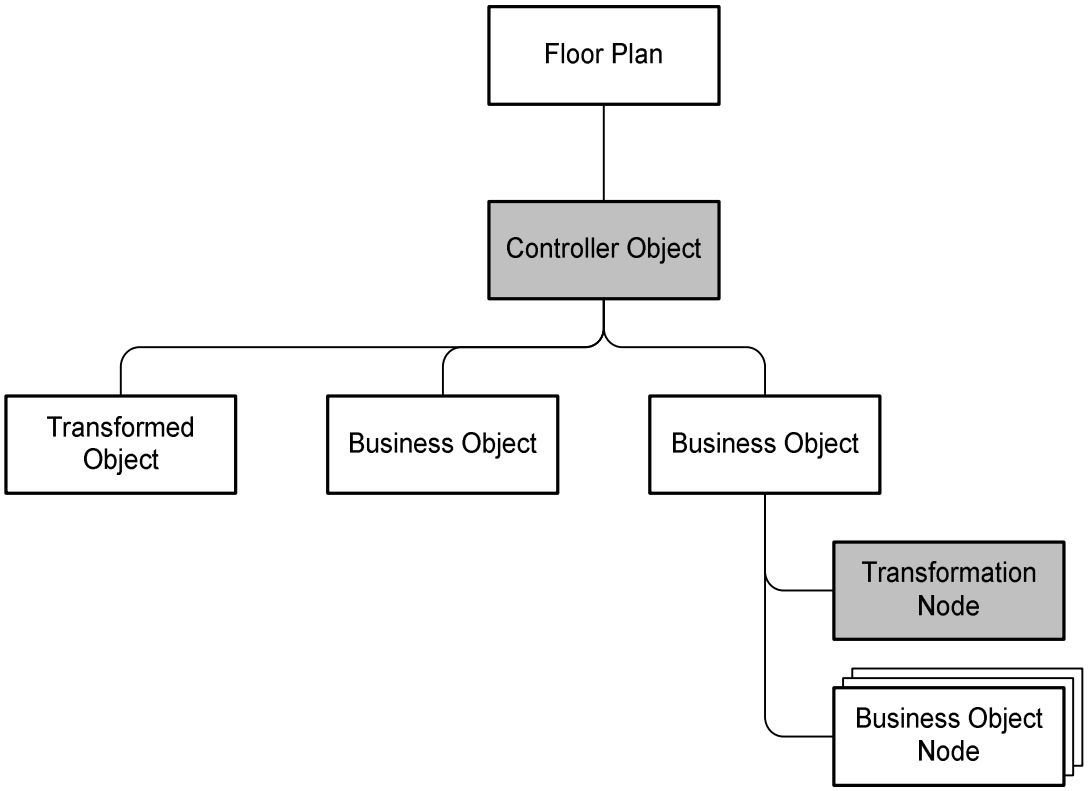


Figure 6-16 Binding a Floor Plan to a Controller Object

6.3.2.4 Service Adaptation with SAP NetWeaver Visual Composer

The service adaptation functionality provided by SAP NetWeaver Visual Composer enables developers to flexibly combine data from multiple business object nodes.

When modeling the user interface, the developer first binds a business object to a floorplan by selecting a certain node of the business object. Afterwards he can enrich the user interface with data from nodes associated to the selected node by using SAP NetWeaver Visual Composer service adaptation. At runtime, the portal application generated from the Visual Composer model calls the core services of all business object nodes added via this technique.

6.4 Further User Productivity Improvements

The AP / ByDesign UI is continually advanced to improve user productivity. Latest development projects are in the areas of groupware integration and Web 2.0.

Groupware integration allows the interaction between AP / ByDesign and mail clients such as MS Outlook. Web 2.0 supports rich internet applications, among others by mashups and syndicated content embedding.

6.4.1 Groupware Integration

Groupware integration of AP / ByDesign is not restricted to a fix set of groupware applications. Instead AP / ByDesign is able to interact with all groupware servers that use standard POP3/IMAP and SMTP protocols. This is achieved by a integration strategy that is completely based on e-mails. Contacts (address data) are transferred using the common vCard format, appointments are exchanged using iCalendar format.

According to the AP / ByDesign process integration strategy, the interaction with groupware servers takes place using compound services and process agents. These process agents receive or assemble SOAP messages according to specific message data types defined for groupware integration. There is one message data type defined for e-mail messages, one for groupware appointments, one for groupware tasks, and one for groupware contacts. Outbound process agents assemble SOAP messages according to one of these message data types and send them to the groupware server. Inbound process agents receive messages from the groupware server in the corresponding format as well. The groupware adapter of SAP NetWeaver Exchange Infrastructure provides handlers to execute necessary mappings and transformations between e-mail types (message, vCard, iCalendar) and corresponding SOAP message types, as shown in figure 6-17).

Users can directly launch the AP / ByDesign user interfaces from the groupware client using URL navigation.

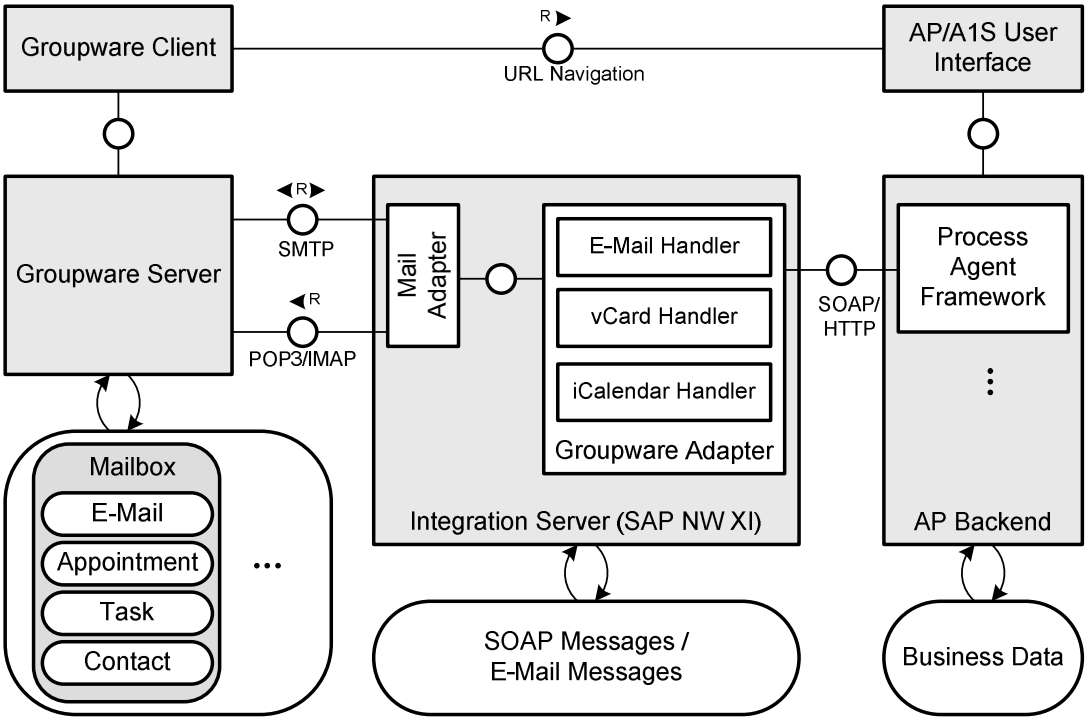


Figure 6-17 Groupware Integration in AP / ByDesign

6.4.2 Web 2.0 Mashups and Syndication

The seamless integration of services and content from various providers in one user interface is a key element of Web 2.0 and enables new ways of user interaction.

Combining and linking multiple services and content in one user interface is referred to as mashup, for example showing a map to locate a customer based on his address record. Typical service providers are Google Maps or Yellow Pages.

Syndication services provide content (Web feeds) that can be integrated in other Web sites or be aggregated by desktop clients. Typical syndication services are Reuters (news) or Yahoo! Finance.

The AP / ByDesign user interface supports the following Web 2.0 scenarios:

- Integrating third-party Web services (mashups)
- Embedding syndicated Web content

In the first case, the AP / ByDesign user interface calls external Web services using parameters derived from other displayed content, such as address fields, and displays the resulting content.

In the second scenario, the AP / ByDesign UI embeds Web feeds offered by syndication services using the really simple syndication (RSS) protocol.

In the central configuration system, customers can define for each user interface the desired Web services and RSS Web feeds.

6.5 In a Nutshell

In order to optimize the organization of an end user's work and to improve an end user's productivity, the AP / ByDesign UI is structured in control centers, work centers, and application UIs.

SAP NetWeaver Visual Composer is the tool for modeling the user interface, using standardized UI building blocks (floor plans, UI patterns, and UI elements), which are built primarily with Web Dynpro for Java. SAP NetWeaver Portal provides the overall infrastructure for accessing the user interface.

The strict decoupling of UI logic from business logic allows the application developer to model multiple user interfaces for a given business object, thereby addressing the needs of different user groups.

User interfaces can be adapted easily to changing business needs due to the pattern-based and model-based UI development approach as well as standardized access to business data via services.

With groupware and Web 2.0 integration, the productivity of AP / ByDesign users can be further improved. All groupware servers compliant to POP3/IMAP and SMTP are supported. Web 2.0 integration lays the foundation for a new genre of interactive mashup applications and RSS-based embedding of syndicated Web content.

7 Supporting Analytics

Analytics is about providing users with information they need for business process monitoring, tactical as well as strategic decision making, and business planning. With regard to analytics, we can distinguish between analytical reporting and operational (operative) reporting. Analytical reporting supports strategic decision making and business planning, while operational reporting is used for monitoring business processes and making tactical real-time decisions.

Two principal categories of data are exploited by analytics:

- Real-time business data as maintained by transactional applications, also known as online transactional processing (OLTP) data
- Aggregated and replicated business data, stored separately and prepared specifically for complex queries, also known as online analytical processing (OLAP) data

Operational reporting is based on real-time data of the OLTP system, while analytical reporting exploits OLAP data.

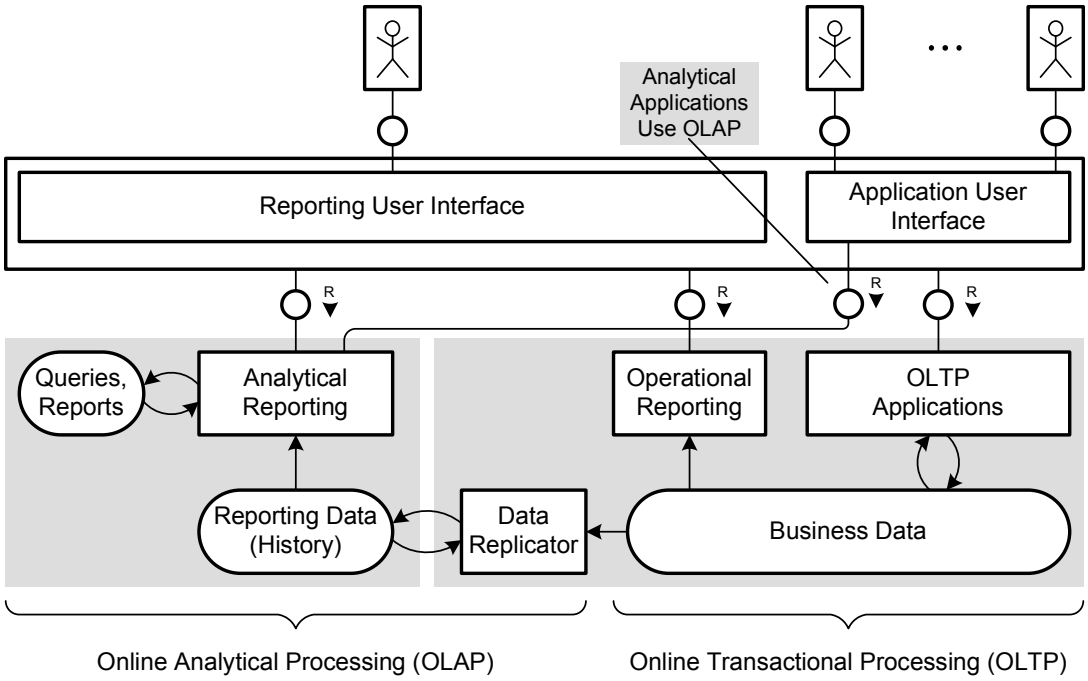


Figure 7-1 Analytical and Operational Reporting

In the past, OLAP typically used historical data with high latency between transactional and replicated data. Today, it is possible to minimize this latency to nearly real time (see section 7.1.4.2. and 7.1.4.3). With this, analytical reporting can guide transactional application users by providing up-to-date key figures to make current business situations transparent. This type of analytical reporting is referred to as process-centric analytics. When integrated with transactional applications in a single user interface, we talk about embedded analytics (see chapter 6.1.3).

In AP / ByDesign, analytical reporting is based on the SAP NetWeaver® Business Intelligence component (SAP NetWeaver BI), which is a full-fledged data warehouse complemented by advanced analytical functionality allowing near real-time analytical reporting (see section 7.1). Operational reporting is based mainly on the SAP fast search infrastructure combined with the SAP search engine (TREX) (see section 7.2). Both analytical and operational reporting follow a model-driven, declarative approach, thus minimizing the effort to compose customer-specific analytics.

7.1 Analytical Reporting

To enable process-centric analytics, specific data replication and access mechanisms feed SAP NetWeaver BI with the necessary real-time data. SAP NetWeaver BI supports analytical reporting through analytical Web applications, which can be embedded as iViews into control centers and work centers (see chapter 6.1.3 and 6.2).

The basic architecture of SAP NetWeaver BI is described in the following sections. After that, we explain the approaches for data extraction from OLTP business objects.

7.1.1 Basic Architecture of SAP NetWeaver Business Intelligence

The basic architecture of SAP NetWeaver BI is depicted in figure 7-2. Like typical data warehouses, SAP NetWeaver BI has its own storage for replicated data and an analytical engine for executing queries and analytical calculations. For storing and presenting data in a way that is optimized for all kinds of reporting, SAP NetWeaver BI uses specific BI data objects called info providers. The most important types of info providers are:

- Info cubes
  An info cube provides a multidimensional view on a set of data and is represented by a number of relational database tables. Info cubes store historical data and are the standard info providers for analytical reporting.
- Data store objects
  Data store objects store replicated OLTP data and contain a change log. They provide delta updates for info cubes.
- Multiprovider
  A multiprovider combines data from several info providers into a new view at runtime. An example of a multiprovider is shown in figure 7-4.
- Hybrid provider
  A hybrid provider merges historical data with new delta information for real-time analytics.
- Virtual provider
  A virtual provider directly accesses OLTP data on demand to create snapshots of data. These snapshots are transient, that is, this data is not persisted (see section 7.1.4.3).

OLTP data, which has to be extracted and replicated to the data warehouse, is defined in the OLTP system as data sources (see section 7.1.3). Data extraction from the OLTP system is performed by BI agents, which are part of the process agent framework (see chapter 5.1.2 and section 7.1.4). After replication to SAP NetWeaver BI, the data is stored initially in the persistent staging area (PSA). From there, it is transformed for analytical purposes and stored physically in info providers.

Queries on info providers are performed by the SAP NetWeaver BI analytical engine. The analytical engine delivers its results via BI consumer services (BICS). Reporting user interfaces, such as BI Web applications, call BI consumer services and present the query results.

If the info provider is an info cube, the BI accelerator can be used to enable fast query processing with low administration effort. The BI accelerator is a high-performance analytics appliance based on the SAP search engine and a specific hardware architecture using scalable blade servers [EFK+06].

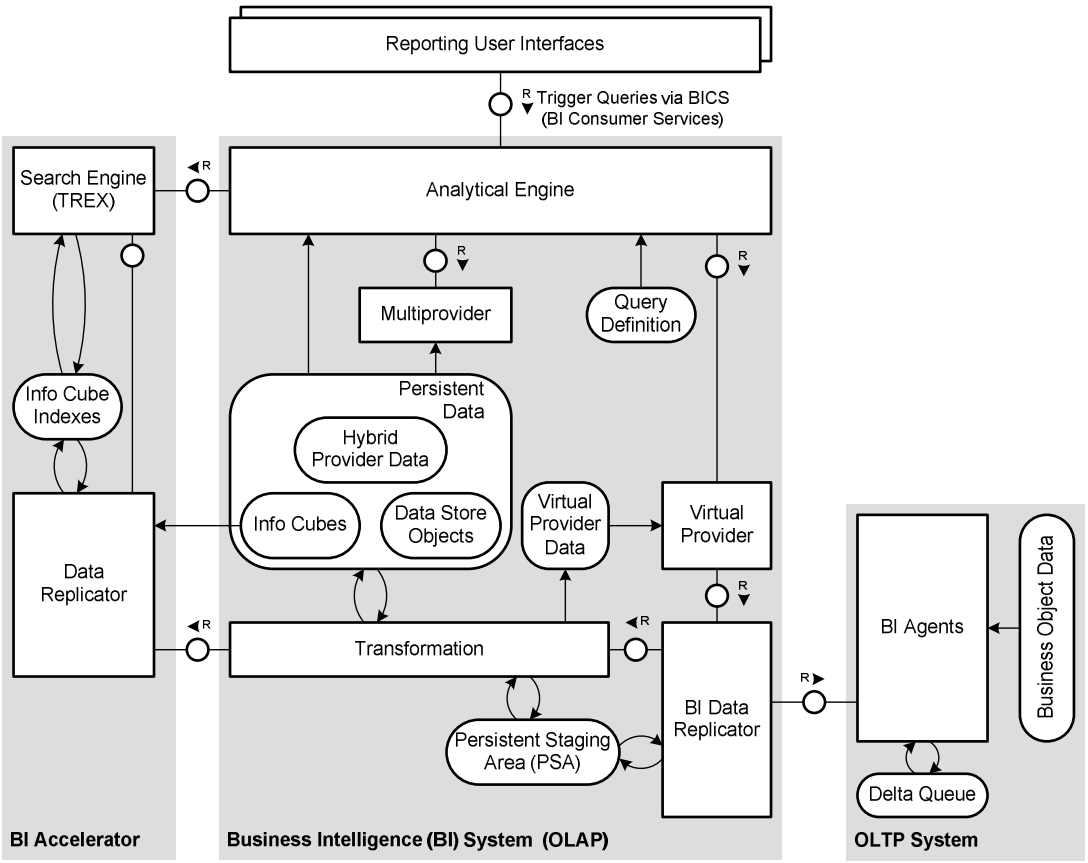


Figure 7-2 Basic Architecture of SAP NetWeaver BI

7.1.2 Data Access Scenarios

In AP / ByDesign, we differentiate between three data access scenarios using three different types of info providers:

- Access to replicated data stored in the data warehouse using info cubes and data store objects
- Access to historical data merged with new delta information for real-time analytics using hybrid providers
- Direct access to OLTP data using virtual providers

The first two scenarios are used for analytical reporting. The last one, direct access, is applicable only for use cases of operational reporting that require the analytical functionality of SAP NetWeaver BI (for example, for legal and compliance reporting purposes). Operational reporting typically uses the fast search infrastructure as described in section 7.2.1.

7.1.3 Business Objects and Data Sources

Business objects of the OLTP system provide the data required for analytics. However, for analytical purposes, it is neither useful nor fast to extract complete business object instances from the OLTP database to SAP NetWeaver BI for the following reasons:

- Typically, only a limited number of a business object's attributes are relevant for analytical purposes.
- Analytics requires additional data, such as calculated key figures based on business object data.
- In many cases, business object attributes must be transformed into an analytics-relevant representation.
- Analytical reporting typically works on a combination of multiple business objects instead of one single object.

To avoid extracting complete business object instances, data sources (see section 7.1.1) are used to describe the required subsets of business object attributes. A data source comprises the following information:

- Extraction structure, which defines how extracted business object attributes are arranged in extraction records (flat data structures)
- Field-to-field mapping between business object and extraction structure
- Name of BI agent used for extraction (see section 7.1.4)
- Application call-back classes, which can be used for data-source-specific mappings, calculations, and aggregations of business object attributes

Data sources are modeled in the OLTP system and then replicated to SAP NetWeaver BI. The modeling (also referred to as data source definition) is done using the data source definition builder of the OLTP system and starts with the business object model as defined in enterprise services repository (ES Repository). For each data source, one or more info packages are defined in SAP NetWeaver BI to control the replication of data from OLTP to the data warehouse. Info packages specify, for instance, the data source origin, mapping rules, and the upload mode (delta or initial).

Figure 7-3 shows the relation between data source, info package, and extraction record.

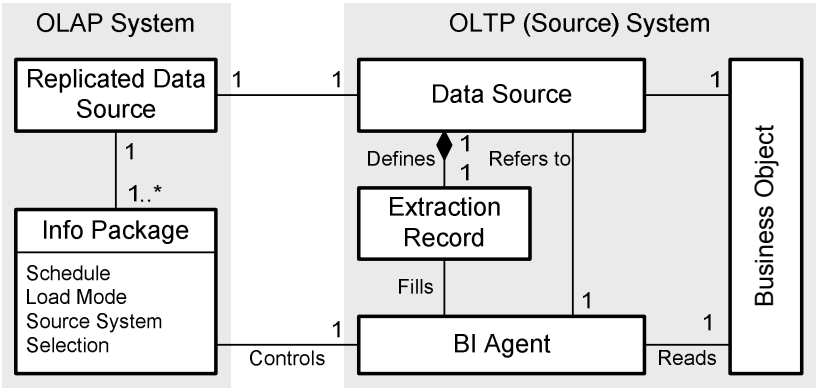


Figure 7-3 Relation between Data Source, Info Package, and Extraction Record

The interplay of business objects, data sources, and info providers is explained in the following example and in figure 7-4:

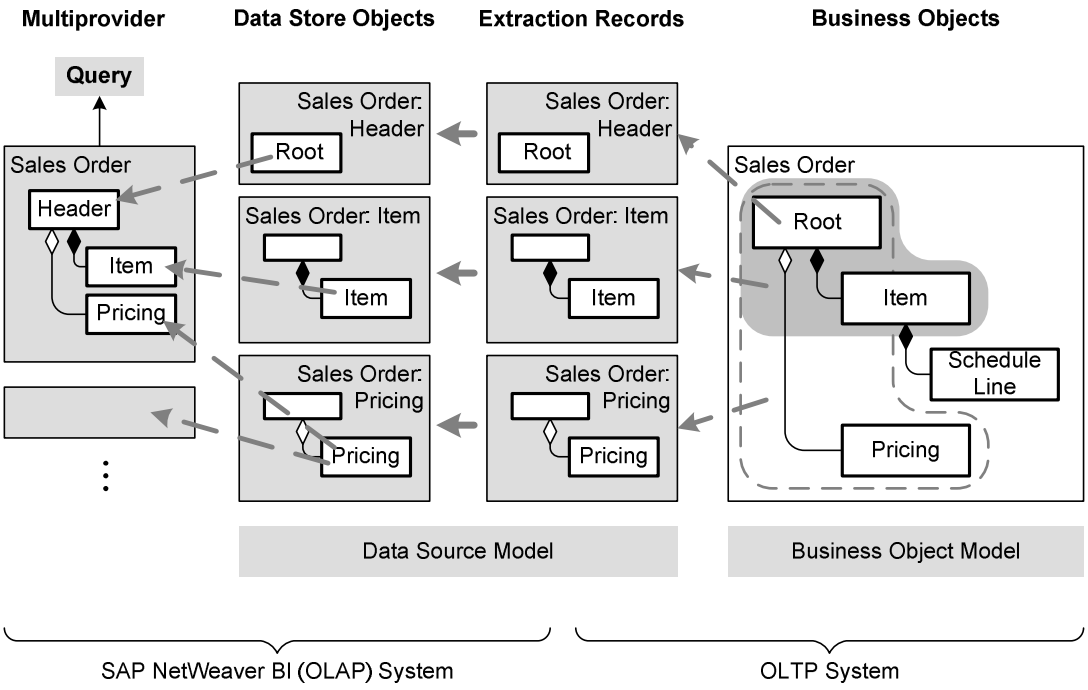


Figure 7-4 Sales Order as Data Source

Example: Sales Order Analysis

Consider a request to develop a set of reports for sales order analysis. To accomplish this, we need to extract the relevant information from the business object SALES ORDER. Data sources are defined for the data that has to be extracted from the business object nodes. In our case, we define a data source for SALES ORDER HEADER, one for SALES ORDER ITEM, and one for SALES

ORDER PRICING (see figure 7-4). These data sources contain the full data set that is required for generation of reports for sales order analysis. The data sources are uploaded into three data store objects in SAP NetWeaver BI. At runtime, data of the three data store objects is combined in one multiprovider, which handles queries defined for it.

The distribution of data across multiple info providers allows the flexible combination of these with other info providers for different reporting purposes.

7.1.4 Data Extraction

Because extracting data for analytics and assembling messages for process integration work similarly, the process agent framework (see chapter 5.1.2) is used for both tasks. For analytics, there are two dedicated types of agents:

- Generic BI agents, which extract and forward data as defined in the data source to SAP NetWeaver BI without any modifications
- Specific BI agents, which allow the implementation of data extraction and transformation for specific use cases (for example, direct access)

In the normal case, the generic BI agent is used to handle data extraction. Unless a specific BI agent is defined for a data source, the BI agent controller instantiates at runtime a generic BI agent for each affected data source. BI agents extract the business object's data according to the update mode (initial or delta), as defined in the corresponding info package (see figure 7-5). Details of the various update modes are explained in the following sections. BI data source services handle the data transfer and communication between SAP NetWeaver BI and OLTP.

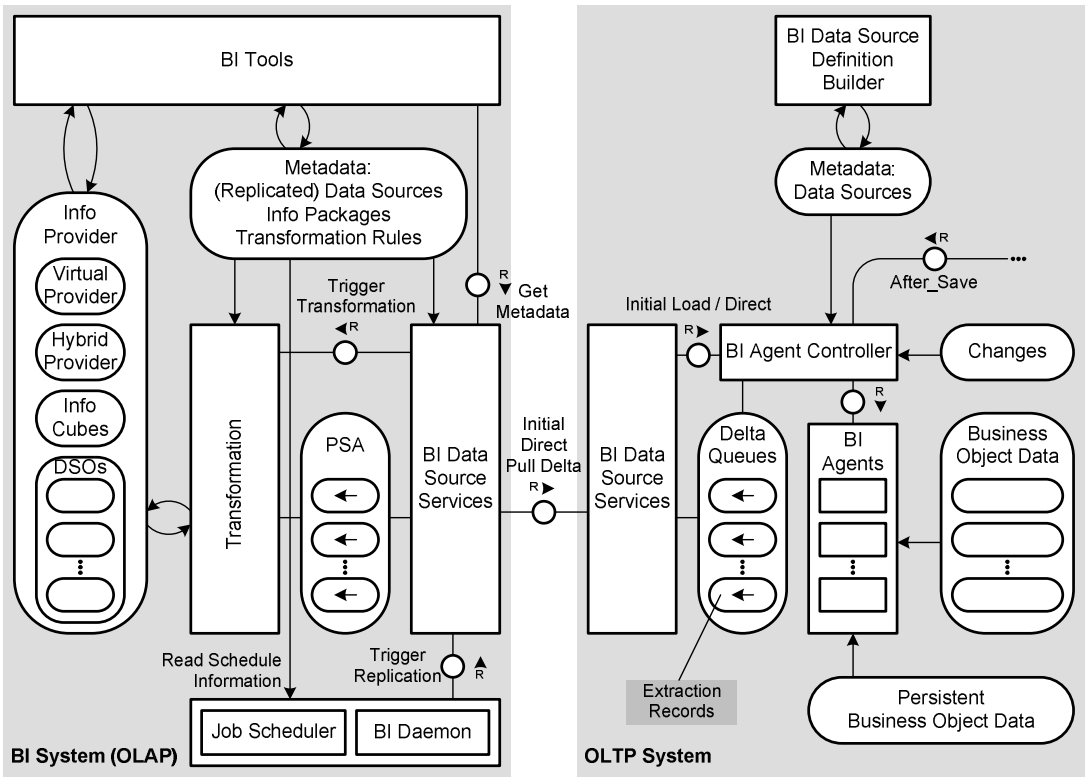


Figure 7-5 Data Extraction via BI Agent

7.1.4.1 Initial Load

For each info provider with its own persistency, an initial load is needed to import existing data from the OLTP system to SAP NetWeaver BI. SAP NetWeaver BI triggers this initial load as specified in the related info package. The BI agent reads the data of all instances of its related business object from the OLTP data store. The data is mapped to the extraction records of the corresponding data source and replicated to the persistent staging area (PSA). From there, it is forwarded to the corresponding data store objects and info cubes.

After the initial load is finished, only data changes are replicated to SAP NetWeaver BI. This mechanism is called delta load.

7.1.4.2 Delta Load

Delta load is initiated when a business object is changed. The BI agent controller instantiates a BI agent for each data source that is affected by the business object change. For each data source, the controller passes the node IDs of all changed business object nodes to the corresponding BI agent. The BI agent retrieves the

required business object data from main memory and writes the values of changed attributes or information about deletion to the extraction records. Afterwards, the BI agent controller passes the extraction records to a delta queue.

Two mechanisms are implemented to pull data from the delta queue:

- Asynchronous data transfer via job scheduler
- Synchronous data transfer via BI daemon

Asynchronous calls to the delta queue are initiated by the job scheduler and are triggered by certain start conditions. The extracted data is stored temporarily in the PSA before being transformed and routed to data store objects.

Synchronous calls to the delta queue are required to support real-time analytics (see section 7.1.5) and are initiated by the BI daemon. The BI daemon triggers the replication of extraction records from the delta queue to SAP NetWeaver BI at regular intervals, for instance every 20 seconds. The extracted data can either be stored in PSA or routed directly to data store objects.

To ensure that the BI agent retrieves the business object data in a consistent state, the transaction schema (introduced in chapter 4.5.2) is enhanced by the AFTER SAVE phase. AFTER SAVE is initiated by enterprise services infrastructure (ESI) directly after the *save* method of the service provider class is called successfully. At AFTER SAVE, the BI agent controller instantiates a BI agent for each data source of each changed business object to perform delta loads.

7.1.4.3 Direct Access

If necessary, queries can access the OLTP data store directly. In this case, a query uses a virtual provider (see section 7.1.1) to trigger the BI agent controller directly by circumventing the replication mechanisms. The BI agent controller calls a specific BI agent that retrieves the required data from the database into the extraction records. The BI agent controller passes this data to the BI data source services, which transfer it to SAP NetWeaver BI. After the necessary transformations and post-filtering, the data is forwarded to the query (see figure 7-5). In contrast to replication, data accessed directly from the OLTP system is not persisted by OLAP.

7.1.5 Hybrid Approach for Real-Time Analytics

To enable process-centric analytical reporting based on real-time data, data is extracted, transformed, and loaded using a hybrid approach. To reduce data latency close to real time, historic data already available in the data warehouse is combined with delta updates requested most recently by the BI daemon.

The hybrid approach is implemented using a new type of info provider: the hybrid provider. When a query is performed, the hybrid provider executes two read accesses in parallel and merges the result. Historical data is retrieved from an info cube. The delta-to-real-time data is taken from the change log of a data store object. To provide the data store object with the latest delta update, the BI daemon pulls data from the OLTP delta queue with high frequency (see section 7.1.4.2).

To minimize the query's response time, the info cube can be indexed by BI accelerator (see section 7.1.1).

7.2 Operational Reporting

Operational reporting includes line-item reporting, user-related object work lists, and explicit object searches as supported for single object maintenance. As an example, figure 7-6 shows an object work list of purchase order instances.

Operational reporting deals with OLTP data instead of data replicated to OLAP. Therefore, in AP / ByDesign, queries for operational reporting are performed by query core services or, if the modeling flexibility of the SAP NetWeaver BI query designer is required, by using direct data access via virtual provider (see section 7.1.4.3).

Query core services can be implemented manually in the service provider class as described in chapter 4.2.1. A better alternative is exploiting the generic functionality of the fast search infrastructure, described in the following section.

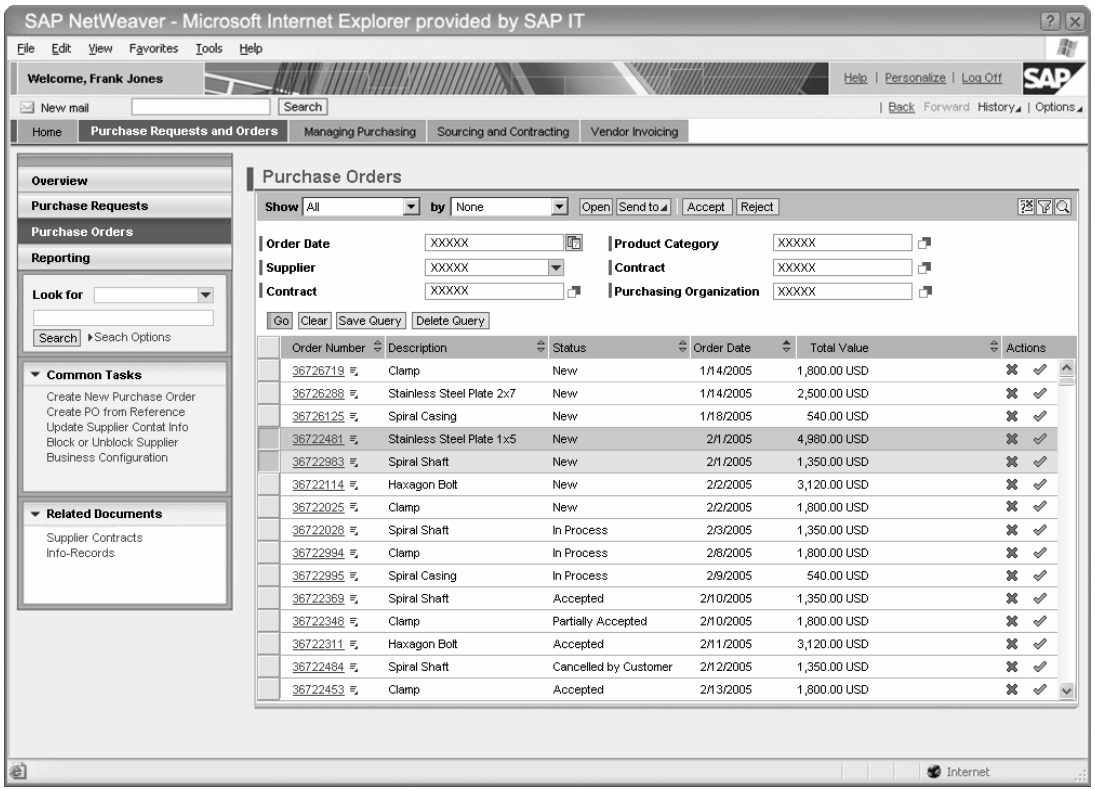


Figure 7-6 Object Work List for Purchase Order Instances

7.2.1 Fast Search Infrastructure

Instead of implementing query core services directly in the service provider class, the fast search infrastructure (FSI) can be used. FSI uses an external search engine to provide very short response times for generic execution of query core services on business objects. This approach also supports special search features such as freestyle search.

The SAP search engine includes a secondary persistency to store and index business object nodes in an optimized way for query execution. The secondary storage decouples queries on business objects from the OLTP data store, which ensures that query execution has no impact on the performance of OLTP.

The runtime architecture of FSI consists of the following components (see figure 7-7):

- SAP search engine with secondary data store
- Replication framework
- Generic query service provider

The FSI replication framework replicates the required business object data automatically (initial load and updates) to the SAP search engine storage. FSI is registered on the event BUSINESS OBJECT CHANGED. When a business object instance is created or modified, the FSI replication framework is informed by ESI during the transactional phase AFTER_SAVE and replicates the corresponding data.

For the design, the FSI query and view builder allows to create query core services by declaring an FSI query and the corresponding view. The view describes the subset of business object attributes that can be presented as result. FSI replicates only those attributes to the secondary data store that are part of the view.

At runtime, ESI calls the generic query service provider that uses the query definition and generates the selection statements that are sent to and executed by the SAP search engine.

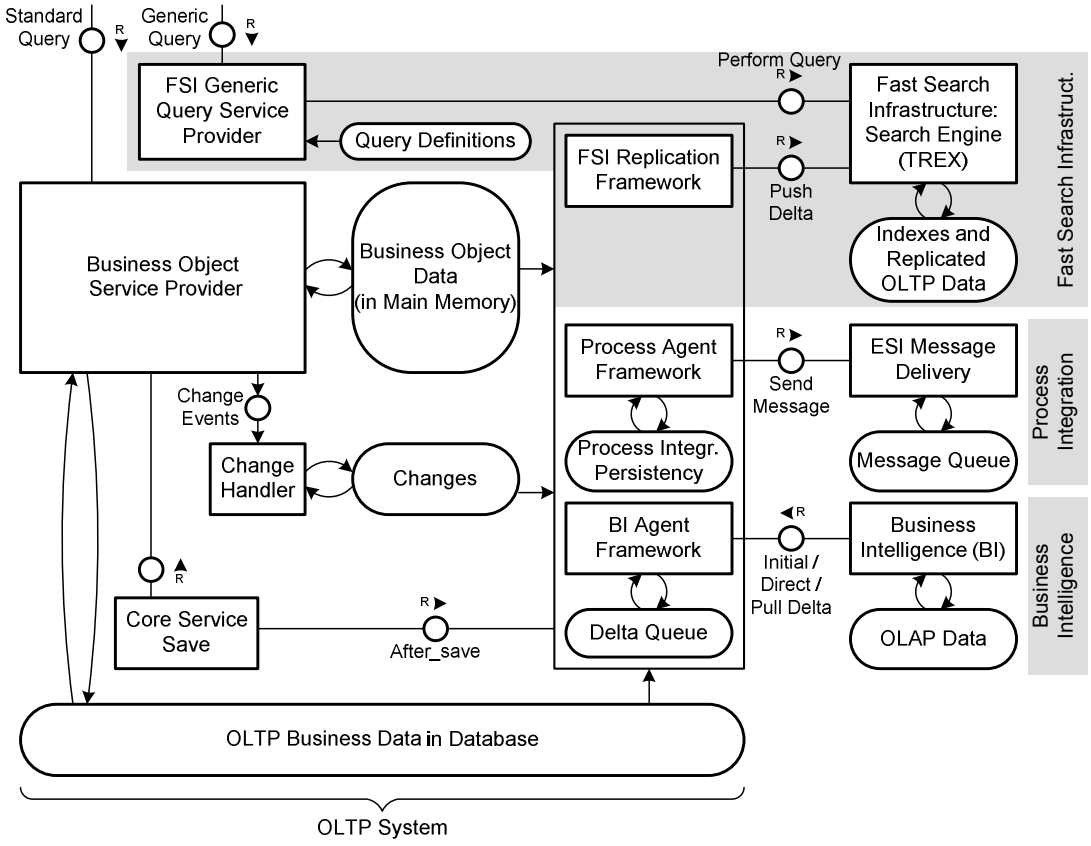


Figure 7-7 Fast Search Infrastructure Architecture

7.3 In a Nutshell

AP / ByDesign analytics provides users with a comprehensive set of analytical and operational reports supporting decision making, planning, and business processes monitoring. Analytics is embedded in users' control centers and work centers using standardized UI building blocks.

SAP NetWeaver BI provides the infrastructure for analytical reporting. The BI accelerator, which is based on the SAP search engine (TREX), enables extremely effective interactive analyses.

Business object data relevant for analytics is collected and transferred to SAP NetWeaver BI using the process agent framework. The data to be extracted is defined by data sources that can be modeled for each business object. BI agents are actively triggered either by requests from SAP NetWeaver BI or by change events that are raised when a business object has been modified.

Operational reporting is based on business object data accessed via core services. Object search is supported by the fast search infrastructure, including the SAP search engine.

8 Enabling Business Process Flexibility

Business applications have to be adapted to customer-specific business processes for the following reasons:

- Set up the system according to the customer's business
- Reflect changing business needs (for example, resulting from legal changes, mergers, or acquisitions)
- Differentiate from the competition and offer unique selling points
- Achieve regulatory compliance

This demand for adaptation exists not only during the first implementation of an application, but throughout its entire life cycle. AP / ByDesign enables business process flexibility in the following ways:

- Role-specific adaptation of the user interface (see chapter 6)
- Business-driven application adaptation (formerly known as business configuration)
- Extension of existing data structures and business functionality
- Creation of new composite applications

Business-driven application adaptation provides the possibility to select certain options to run a business application or to fine-tune predefined settings. The application itself delivers these configuration possibilities (see section 8.1).

Extensions enable the enhancement of business functionality and data structures beyond what is delivered by the software provider. AP / ByDesign provides specific tools and features for extensions (see section 8.2).

Composite applications come into play if additional business processes need to be supported. Composite applications use data and functions provided as services by existing applications and combine these with their own business logic and specific user interfaces (see chapter 2.2.3 and section 8.3).

8.1 Business-Driven Application Adaptation

The business-driven application adaptatation concept of AP / ByDesign starts early, at the sales phase, when the customer is evaluating the software product. It is based on the idea to narrow the scope of AP / ByDesign to match the customer's requirements. The customer gets access to a trial system with predefined configuration settings that serve as the basis for all subsequent refining configuration steps. At the end of the process, configuration settings are deployed to the customer's production system.

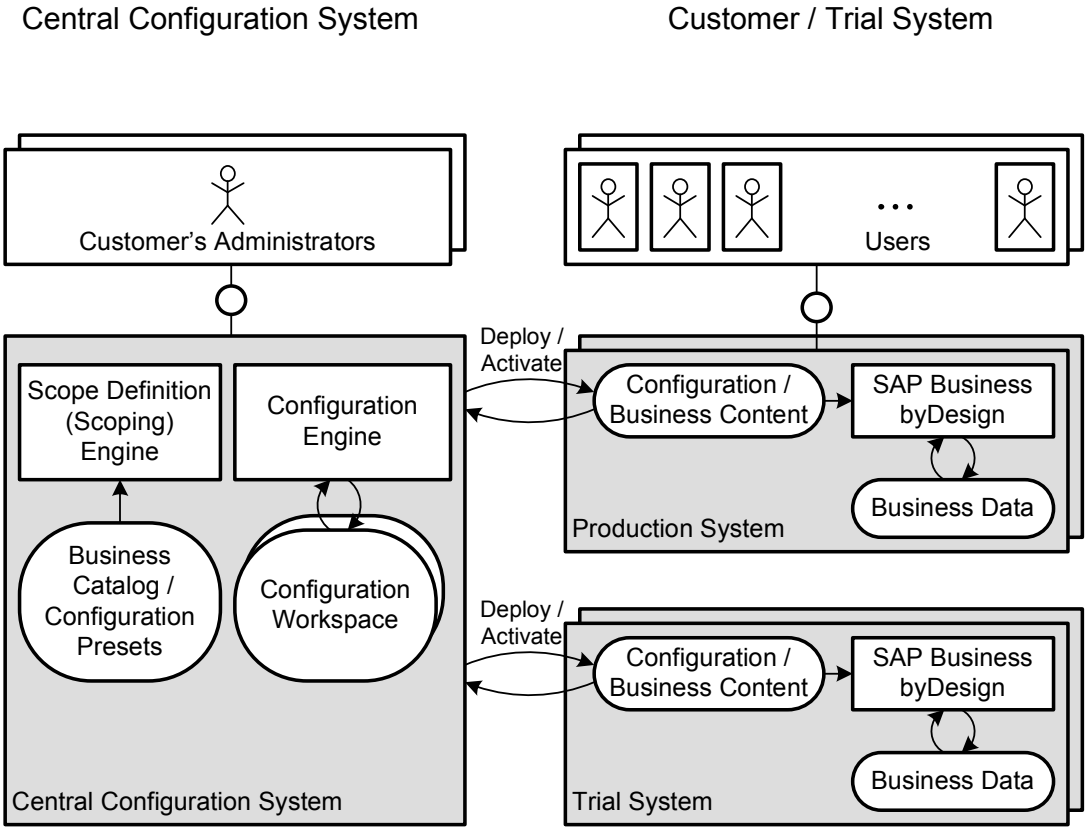


Figure 8-1 Configuration Landscape

From a technical point of view, the business-driven application adapation is realized by using the SAP central configuration system (also referred to as the central design time and scoping system), which can supply any AP / ByDesign system with configurations. Numerous sets of predefined country-specific and industry-specific configuration settings simplify the configuration process. Companies select and maintain configurations in the central configuration system and replicate them to their own AP / ByDesign installation. During runtime, each AP / ByDesign system uses its

own configuration settings. In this way, the configuration process is decoupled from the customers' systems, thus simplifying consistency and compatibility checks (see section 8.1.2).

The central configuration system includes two engines to support the configuration process (see figure 8-1):

- Scoping engine for selecting appropriate configuration presettings
- Configuration engine for adapting and fine-tuning configurations

Both engines are described in the following sections.

8.1.1 Preconfiguring with the Scoping Engine

Preconfiguring is the process of configuring an initial version of a software application based on standard settings.

The scoping engine of the central configuration system provides a business adaptation catalog that describes and organizes the full set of AP / ByDesign functionality using business language. To select a specific scope, the customer is guided through a set of questions and answers that reduces the options of the business adaptation catalog to those relevant for his industry and countries. The customer continues selecting the specific business functions relevant for his business.

Based on these selections, the scoping engine determines an initial configuration, which consists of business configuration sets and business content:

- Business configuration sets are predefined configuration settings stored in configuration tables. They are provided by the central configuration system and can be replicated to the configuration tables of local systems.
- Business content refers to predefined portal roles (portal content), reports (analytics content), or forms. This content is available in each installed system in an inactive state and needs to be activated if selected.

This configuration can be used to create a trial or an initial production system (see figure 8-1). To do so, business configuration sets are deployed and business content is activated. Afterwards, a configuration can be fine-tuned using the business configuration engine.

8.1.2 Fine-Tuning with the Configuration Engine

When scoping is finished, the selected business configuration sets can be fine-tuned to fit to the customer's specific requirements. For this, the configuration engine of the central configuration system is used.

Fine-tuning of configuration does not deal directly with the configuration tables stored in the AP / ByDesign system. Instead, simple, well-defined configuration views are provided, which abstract from the details of the configuration tables. These views enable quick, easy, and low-cost implementation projects. The settings made in configuration views are later mapped to configuration tables using transformation rules maintained in the metadata repository of the configuration engine (see figure 8-2).

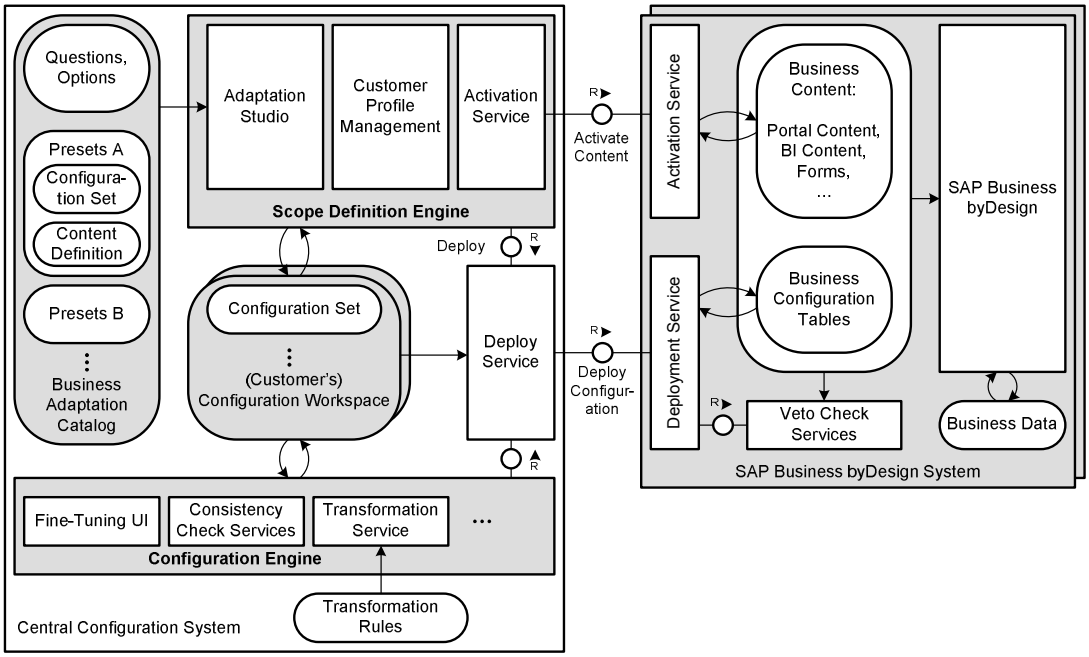


Figure 8-2 Configuration Architecture

On the central configuration system, each customer has its own workspace for storing configuration settings and selections. Consistency checks evaluate whether the newly selected configuration settings are consistent with already existing settings within the customer's configuration workspace. During fine-tuning, the current configuration can be saved at any time as a draft. However, only consistent configuration settings can be deployed to the local systems.

Once the configuration process is completed, the final configuration is deployed to the customer's local system. During deployment, veto checks verify whether the new configuration settings are compatible with already existing business object instances, especially master data, in the local system.

8.2 Extensions

AP / ByDesign allows extending data structures and business processes in a flexible manner while ensuring smooth updates and upgrades to newer software releases. Therefore, all compliant extensions and changes are free of source code modification, which means they are not overwritten by updates and upgrades.

Partners and customers can extend existing AP / ByDesign functionality in the following ways:

- Adding additional fields to existing business object nodes
- Adding additional business logic
- Adding process-related extensions

With AP / ByDesign, extensions can be implemented in separate name spaces. This ensures that extensions from customers, ISVs, and internal SAP development do not interfere with each other.

8.2.1 End-to-End Field Extensibility

End-to-end field extensibility means that industry-specific and company-specific fields can be added to business object nodes. These additional fields, also called extension fields, are propagated automatically to all affected layers of the AP / ByDesign architecture. They are stored in the database, controlled by business logic, accessed via service operations, and displayed on the user interface. In addition, analytics, forms (interactive forms and print forms), and search indices have to be adjusted as well.

The infrastructure for adding extension fields is provided by the easy enhancement workbench. When an extension field is created, easy enhancement workbench performs the following steps to make the field available in the database, the business logic, the service layer, and the user interface (see figure 8-3):

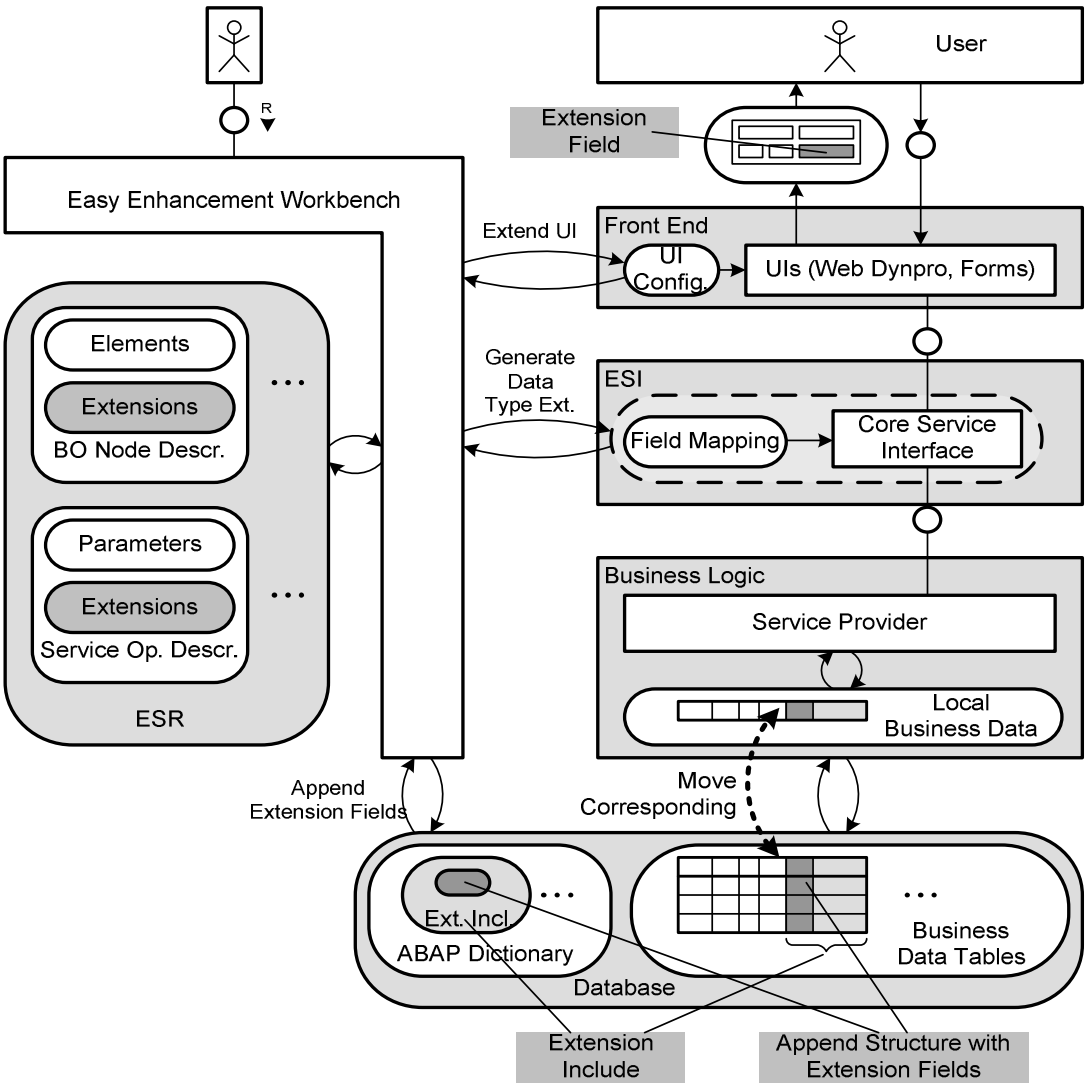


Figure 8-3 End-to-End Field Extensibility

- The field is added to the node data type of the business object node in enterprise services repository (ES Repository). The corresponding core service interfaces are regenerated.
- The field is added to ABAP™ data structures used in the business logic, which also automatically extends the dependent database tables.
- The field is added to SAP NetWeaver® Visual Composer UI models in order to be displayed on the corresponding screens.
- The business logic just passes new fields through; it is not modified.

Business logic in the back end must be prepared carefully to allow easy end-to-end field extensibility. An extension include, which is a data structure that contains only one empty field of one byte, has

to be defined in the ABAP dictionary[5] for each business object node. This extension include has to be added to all data structures used within the corresponding service provider class. Within these data structures, the extension include acts as an anchor for extension fields that are added as append structures.

8.2.2 Enhancing Business Logic

Often it is not enough to extend a business object by additional fields. It might be necessary to change, retrieve, or transform field values. Or there is the need to modify messages assembled by process agents. In both cases, developers have to add ABAP code to the delivered software.

To facilitate this, SAP provides dedicated extension points in the business logic. Developers register their additional code with these extension points. At runtime, whenever the business logic reaches an extension point, it calls all registered extensions before it proceeds. With this mechanism, developers can add custom code without having to modify SAP software. This means that upgrades or updates do not overwrite the added code.

In the special case of extension fields, extension points are typically used for the following:

- Adding check logic for extension fields
- Calculating field values in dependency of other fields or setting a default value for an extension field
- Changing field properties dynamically (for example, read-only, mandatory, and so on)
- Checking user authorization for the extension field

For outbound process agents, extension points allow the enhancement or modification of message content before messages are sent. Corresponding extension points in inbound process agents process the changed messages. In addition, there are dedicated extension points within the business object implementations defined by SAP to allow code enhancements.

[5] The ABAP dictionary centrally describes and manages all data definitions used in the system.

8.2.3 Process-Related Extensions

Besides enhancing business logic, there are several possibilities for process-related extensions of AP / ByDesign that don't require code modification. These extensions can be initiated at different user levels: system administrators, selected (key) users, and end users.

8.2.3.1 Field Extensions and Document Flow

For specific scenarios, it is possible to propagate field extensions along the corresponding business document flow. For example, an automated transfer is provided for the document flow from sales order to customer invoice request and customer invoice. To provide this kind of process-related extensibility, the field extensions of business object nodes are complemented by analogous extensions of the corresponding process agents, messages, and compound service interfaces.

8.2.3.2 Changing the Message Flow

The decoupled nature of asynchronous communication between deployment units offers a convenient way to influence the related message flow by using the integration server of SAP NetWeaver Exchange Infrastructure. Messages can be redirected to other applications or custom-built deployment units. The integration server provides the necessary mapping and routing. It also includes adapters for transforming Web service calls from AP / ByDesign to other communication protocols, such as RosettaNet or remote function calls (RFC).

Currently, process agents cannot be changed without code modification. However, in future AP / ByDesign releases, it will be possible to maintain process agent models within enterprise services repository. By doing so, process steps can be rearranged easily, message choreography can be changed, and additional messages can be created.

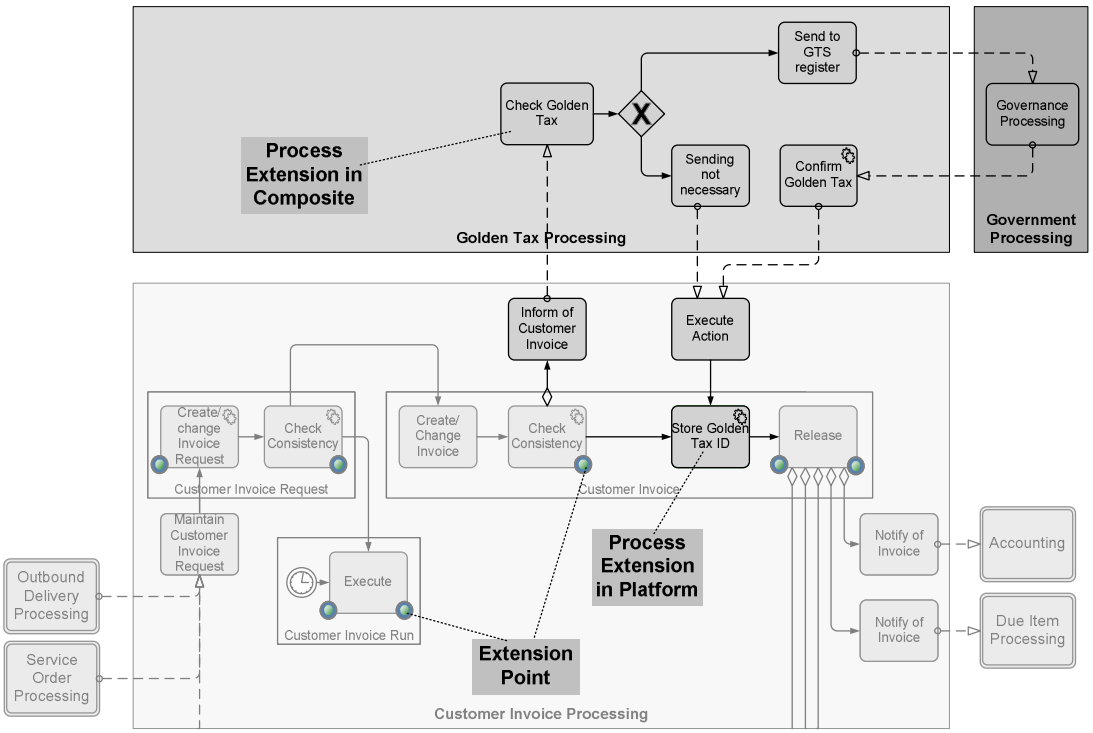


Figure 8-4 Extension of a Business Process

Figure 8-4 shows an excerpt of a process flow model (see also chapter 10.2.1) of customer invoice processing. Some activities provide extension points to append or insert additional process steps, in this example provided by a composite application that requests an individual tax number from Chinese government, which has to be included in every invoice (golden tax processing).

8.2.3.3 Adding and Activating Business Tasks

System administrators handle the process-related extensions we discussed above. In addition, end users can enhance running processes on the fly using business task management, which enables them to address follow-up business tasks to other users. The business object instance the user works on is blocked during such requests. An example is the creation of a business task to ask another user for clarification. As a further option, selected users (sometimes called key users) can activate predefined business tasks, such as an approval, using business configuration as described in section 8.1.2.

8.3 Composite Applications

Composite applications, introduced in chapter 2.2.3, represent the most powerful approach to extend the scope of AP / ByDesign. They combine multiple services from underlying business applications to build new or supplementary functionality.

Composite applications are characterized as follows:

- They are consumers of services and events. They combine services and user interfaces from other providers with their own application logic, user interfaces, and business process orchestration in order to address a specific business requirement.
- They are loosely coupled to the components they are based on. (This is necessary, for example, to ensure compatibility with different releases and configurations of the underlying applications.)
- They are user-oriented applications that support collaboration within and across systems.
- They have their own life cycle and can be built, packaged, deployed, and upgraded independently of the underlying applications by SAP or ISVs.

Composite applications can range in complexity, starting with simple orchestration of existing user interfaces and ending with applications that contain a great deal of their own business logic.

The application platform of AP / ByDesign provides a rich set of highly reusable business objects that can be consumed by composite applications. Composite applications interact with business objects via compound services and process agents in a loosely coupled way. Event-driven process agents make it possible to plug a composite application directly in existing applications.

For developing and running composite applications, SAP provides SAP NetWeaver Composition Environment. It includes a lean version of the SAP Java EE 5 application server, which requires minimal hardware resources and can be installed easily. Model-driven development tools accelerate the creation of composite applications.

8.3.1 Providing Services for Composite Applications

The application platform provides stateless compound services for communication with composite applications. Stateless compound services are much easier to consume, because only semantical, and not technical, dependencies influence the sequence of service calls. Due to the stateless nature of services, one service call corresponds to one back-end logical unit of work (LUW). This ensures that business object instances are not locked over long time intervals by composite applications, thereby preventing other users from modifying them.

Besides compound services for asynchronous application-to-application or business-to-business communication (see chapter 5), application platform provides an additional set of compound services for direct user interface interaction. Composite applications call these services synchronously to get immediate response.

In contrast to compound services, core services of business objects are not intended to be used for composite applications for the following reasons:

- Using core services would allow composite applications to access different deployment units within one LUW. This contradicts the loose coupling of deployment units and could affect data consistency between deployment units.
- The use of core services would give the composite application unwanted influence on memory consumption, locking, and performance of the AP / ByDesign system.

All compound services are published in enterprise services registry, a UDDI-compliant service registry, where they can be browsed by composite application developers.

8.3.2 Developing Composite Applications

From a development perspective, we differentiate between the composite application's user interface, process orchestration, and business logic (see figure 8-5).

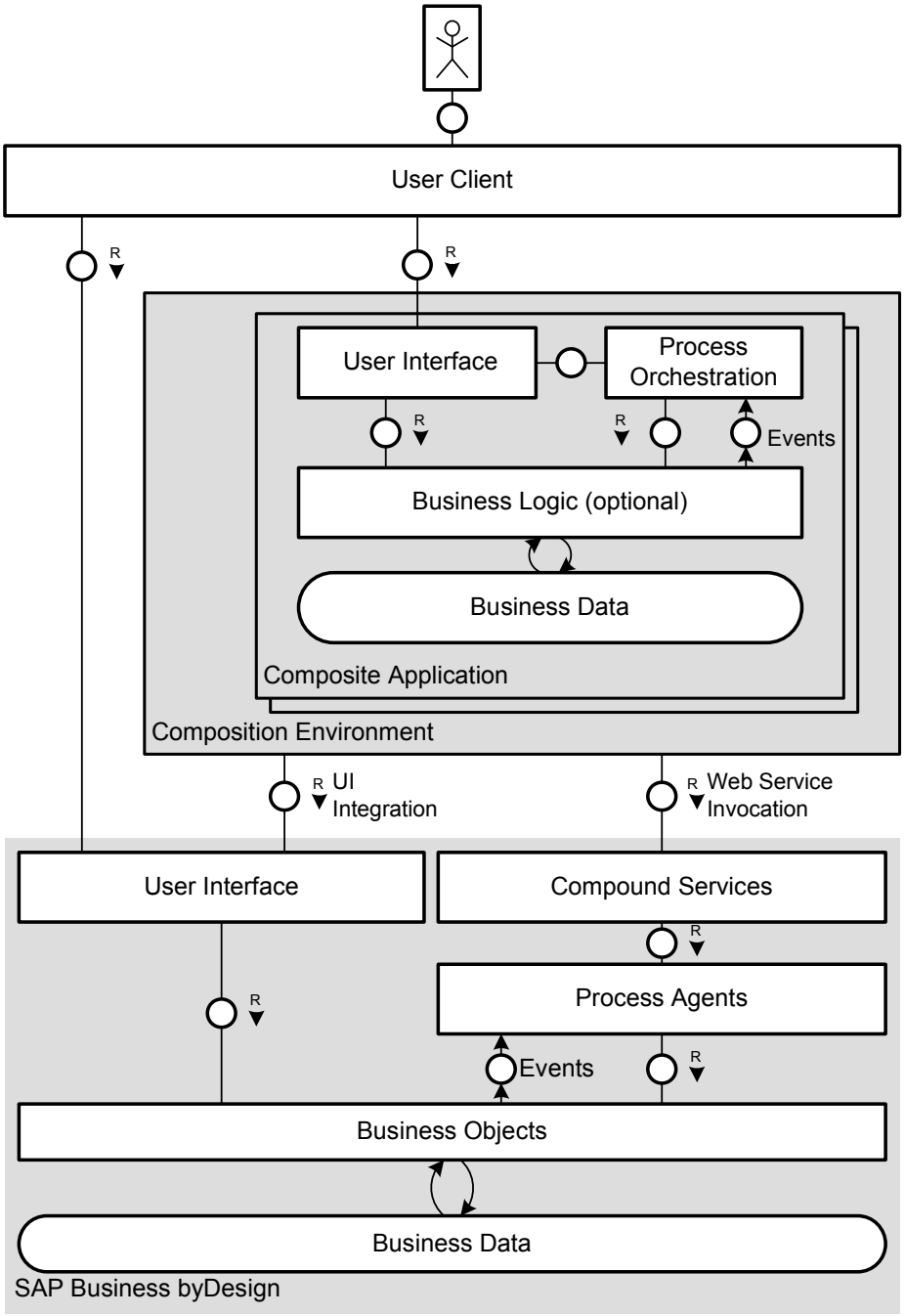


Figure 8-5 Composite Application on top of AP / ByDesign

The composite application's user interfaces are modeled using SAP NetWeaver Visual Composer. The user interface models are based on the compound service definitions from ES Repository and implemented in Web Dynpro for Java. Composite applications can be integrated into existing AP / ByDesign user interfaces.

The composite application's process orchestration defines the flow in which process steps are executed, which user interfaces are presented to which user roles, and how data is passed between

these process steps. Process orchestration is based on the SAP guided procedures technology.

If a composite application requires its own business logic with its own persistency to provide the required functionality, SAP NetWeaver Composition Environment offers a Java EE 5–compliant development environment based on Eclipse.

8.4 In a Nutshell

AP / ByDesign is built in a way that allows companies to flexibly adapt the scope of their solution to their business processes.

Thanks to the central configuration system, customers can get trial systems based on their preferences and requirements early in the sales phase. The following fine-tuning of configuration settings is easy and quick, using configuration views that abstract from the details of the configuration tables. Automatic consistency and veto checks ensure that the deployment of configuration settings does not cause errors and conflicts.

Extension fields that are added to business object nodes are displayed automatically on the user interface and stored in the database. In specific scenarios, extension fields can be transferred along a complete process flow. Dedicated extension points are available to add additional program code in an update- and upgrade-compatible manner. Business task management and SAP NetWeaver Exchange Infrastructure provide a toolset for enhancing and changing business process flows.

Composite applications are the logical consequence of service enablement. They combine multiple services and user interfaces from given software applications to build new or supplementary functionality. The application platform provides stateless compound services for communication with composite applications. With SAP NetWeaver Composition Environment, SAP provides model-driven development tools and a lean Java 5 EE application server for creating and running composite applications.

9 Operating AP / ByDesign

After making the decision for a specific business software solution, enterprises have to start thinking about how to integrate the new solution into their system landscape, how to operate the software, and how to manage the entire system. The objective is to optimize the business value of the software solution while controlling the total cost of ownership (TCO).

Talking about TCO means primarily looking at cost of operation. When considering overall costs, licensing costs for hardware and software normally have a minor impact on a software solution's TCO. With AP / ByDesign, operation costs can be reduced significantly by introducing new concepts for installation, deployment, configuration, operation, and support.

In the following sections we highlight the system and application management concept of AP / ByDesign and examine the mega-tenancy-based hosting solution as a forward-looking approach for operation.

9.1 System and Application Management

The system and application management strategy of AP / ByDesign is based on automated problem-solving and support mechanisms, which are referred to as automated service and support. This approach minimizes manual system and application management efforts and ensure functional reliability by a well-defined service infrastructure that guards the system 24 hours a day. Automated service and support is integrated seamlessly with the back-office system of the support organization – whether internal, partner, or SAP – which guarantees a clear and secure service path based on dedicated service level agreements.

In detail, automated service and support comprise the following:

- Automated system health checks
- Automated support
- Software maintenance management
- Learning content

Automated system health checks and automated support are described in the following sections. Software maintenance manage-

ment pushes software and content updates to AP / ByDesign systems. Learning content is provided to AP / ByDesign users via learning centers, which are embedded in work centers as discussed in chapter 6.1.4.

9.1.1 Automated System Health Checks

Permanent, automated system health checks allow monitoring of AP / ByDesign on a technical and application level to detect any incidents[6] in advance. If a health check detects an incident, system and application management collects all available context information in a software problem report, which is sent to local support via a business-to-business (B2B) message. There a support request is created and forwarded either to local support people, to a partner, or to SAP. The attached context information helps support employees solve the reported problem with minimal communication overhead.

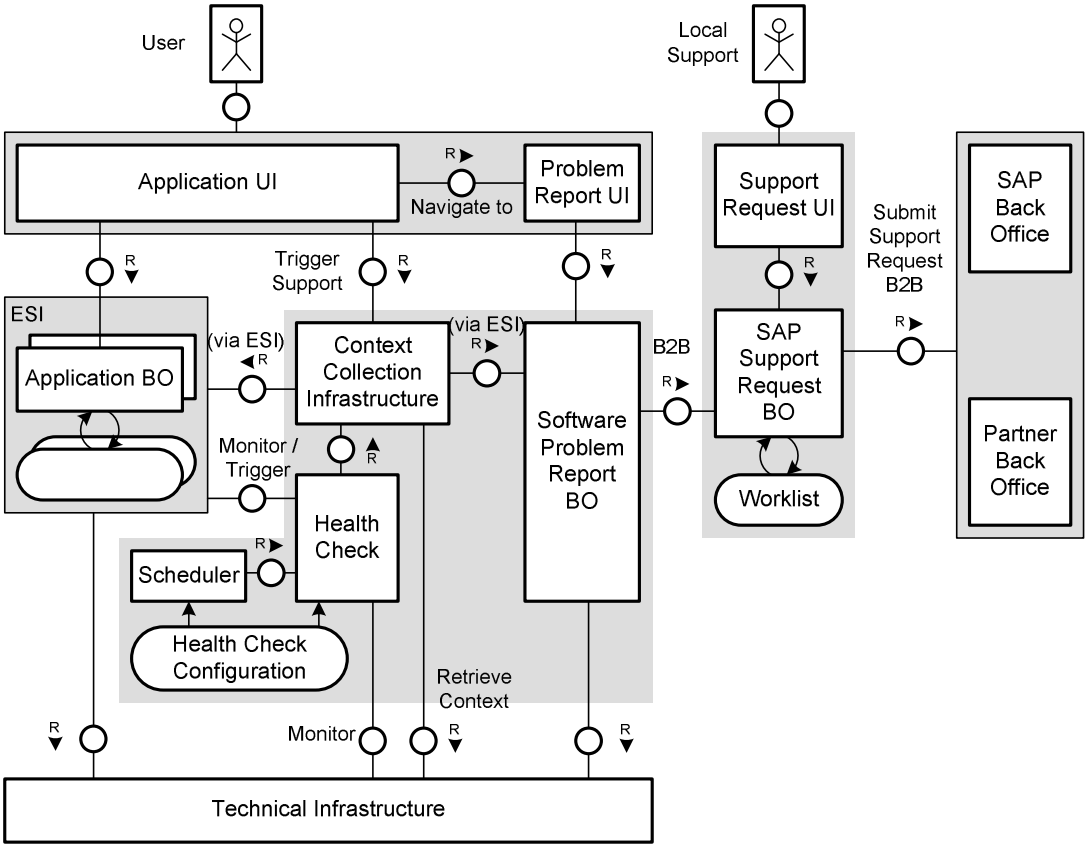


Figure 9-1 Automated System Health Checks and Support

[6] Incidents are any faults, errors, and malfunctions of an IT system.

9.1.2 Automated Support

Automated support guarantees, in particular, the capability of mid-size companies to easily support their software solutions. End users in this market are often not given extensive training, and may use the software sporadically. As a result, the number of support requests generated by these users is high. Automated support enables these users to report incidents from any UI screen, requesting support either directly or by means of the help menu.

In addition to user-initiated requests, support requests can be triggered automatically by critical alerts. For processing the incident, automated support uses the same infrastructure for collecting context information and creating support requests as for automated system health checks (see figure 9-1 and section 9.1.1).

9.2 Megatenancy

For a growing number of midsized companies, it is not cost effective to run their own IT infrastructure, and they prefer to have their IT system hosted by a third party. In addition, new software consumption models are emerging on the market, such as Software as a Service (SaaS), rental software, and so on. Both factors are accelerating the growth of the hosting solutions market. However, this situation is challenging for hosting providers, because their customers are demanding increased flexibility and scalability with reduced TCO for the implementation and operation of a single business application.

Traditional system architectures give hosting providers a choice of two options to set up a hosting system landscape (see figure 9-2). The first, shown on the left of figure 9-2, is a nothing-shared approach, where each customer is provided with a dedicated system. The second, on the right, is an everything-shared approach, where all customers share one system. These two options are also known as a single-system solution versus multiple-client solution. Both options have advantages, but they lack flexibility and do not meet the required speed of change. Translating the key principle of enterprise SOA – loose coupling of reusable components that provide dedicated services – on the IT infrastructure level leads to a third hosting option, which we call mega-tenancy.

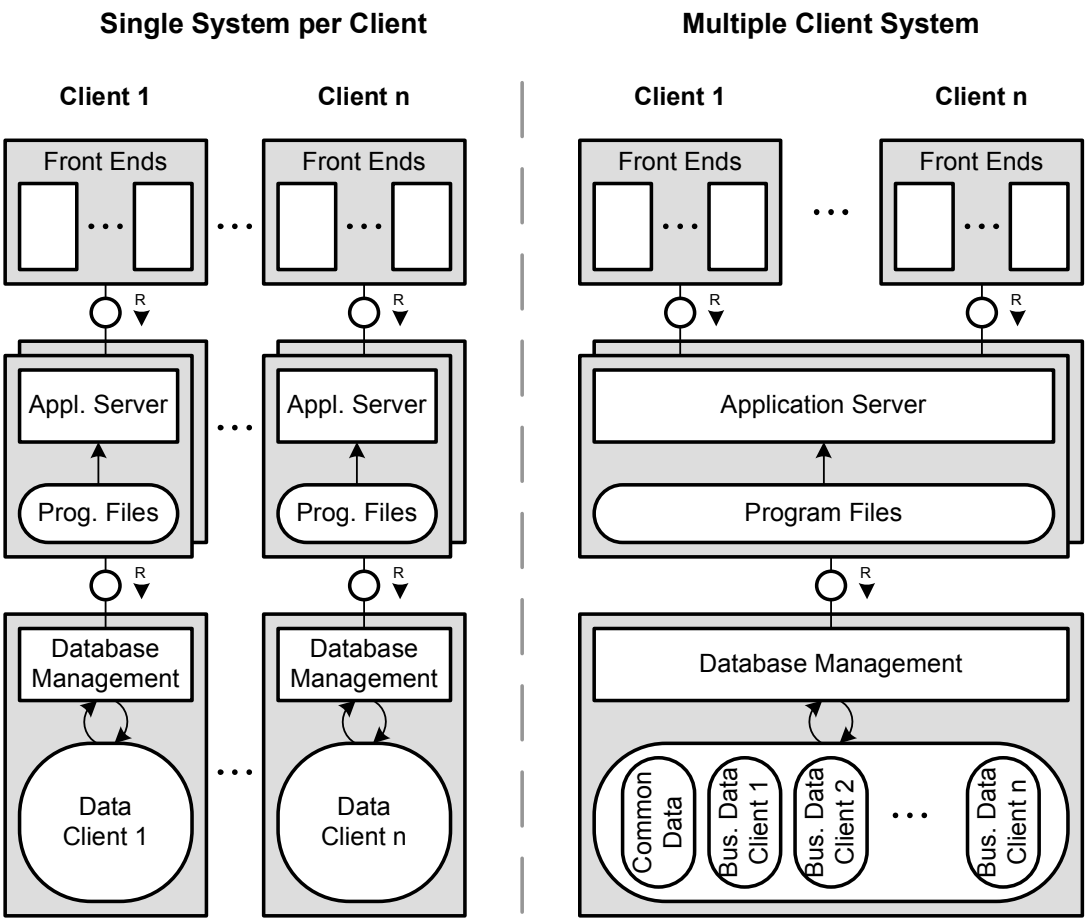


Figure 9-2 Traditional Hosting Variants

Megatenancy combines the sharing of common data and resources with the separation of customer-specific data. Mega- tenancy looks at the IT infrastructure as a set of services that can be leveraged and reused in a flexible and cost-effective way. A prerequisite for megatenancy is the separation of the IT infrastructure into storage, computing, and control.

To implement the megatenancy concept on data storage level, network attached storage is used. Each customer system is represented by a dedicated storage area, called tenant space, for customer-specific business, configuration, and program data (see figure 9-3). Customer-independent data shared across all tenants, such as configuration data, repository data, and ABAP™ programs of AP / ByDesign, is stored only once in the shared space.

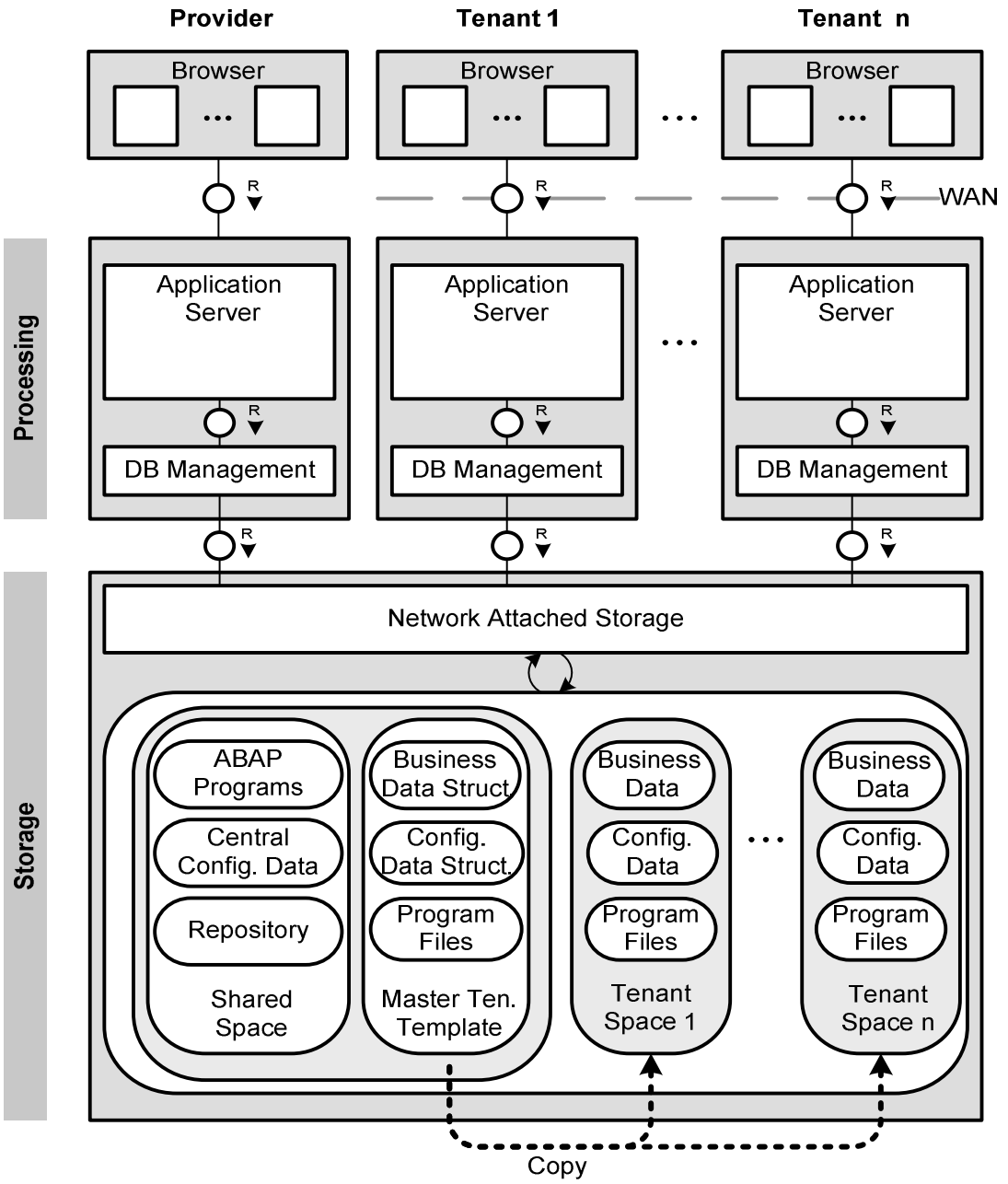


Figure 9-3 Megatenancy Landscape

To set up a megatenancy system landscape, a provider system has to be installed to provide the shared space as well as the master tenant template. The latter contains the data structures needed to store customer-specific business and configuration data as well as the program files of the application server and database management system. Hosting providers create a new tenant for a customer in minutes by copying the master tenant template. No seperate installation is necessary.

At runtime, the application server accesses the customer's own tenant space for customer-specific data and the shared space for

common data. Naturally, customers cannot modify data stored in the shared space (see figure 9-3).

Traditional hosting solutions are often limited by fixed assignment of business applications to IT resources. This tight link is dissolved by megatenancy. SAP's megatenancy approach is completely built on the SAP NetWeaver® platform's adaptive computing functionality, which provides a virtualization layer that allows a flexible assignment of tenant systems to computing resources. For example, the tenant's application server and database management system are not installed on dedicated servers, but are started and executed on idle computing resources. With this, hosting providers can utilize their IT infrastructure to full capacity.

The complete hosting solution is managed centrally using a megatenancy controller for creation, deletion, and monitoring of tenants, and an adaptive computing controller for the flexible assignment of computing resources.

9.3 In a Nutshell

AP / ByDesign system management strategy reduces manual efforts to a minimum and ensures functional reliability by a well-defined service infrastructure that guards the system 24 hours a day.

Automated health checks continuously check the system and proactively inform people in charge about critical availability, scheduling, and performance issues.

Automated support enables end users to request support from any UI screen while providing all necessary context information to help support people solve the problem with minimal effort.

Megatenancy is the first hosting solution that consistently leverages enterprise SOA principles. On one hand, the concept is based on services, which are centrally offered by a central system and shared among multiple tenant systems. On the other hand, megatenancy uses the available IT infrastructure as a set of resources that can be leveraged in a flexible and cost-effective way.

10 Using Model-Driven Development

As introduced in chapter 2.4.1, models are abstractions of (potentially) real things, made to serve one or few purposes, such as clarifying concepts or serving as basis for code skleleton generation.

A key characteristic of the model-driven approach to software development is the specification of different aspects of a system using different languages. Each of those languages can be more adequate to the domain it needs to address, and closer to the concepts in which its users think than any general-purpose programming language. Artifacts are assembled into a final, executable application, often supported, and partially automated by tools, such as model transformers and generators.

The promise of model-driven development is to make the software development process much more effective and efficient, and to create artifacts that are well suited for reuse. To allow models to be the central part of development, it is necessary to have both good tool support and strict compliance to rules so that model and implementation are synchronized. Finally, a central repository has to provide access to all models during all phases.

10.1 Development Phases for AP / ByDesign

Enterprise SOA, in general, and AP / ByDesign, specifically, are closely interlinked with the principle of model-driven development. This chapter covers the model-driven development process of AP / ByDesign and the underlying application platform and discusses the model types and tools used during the various development phases from specification and design to generation and implementation. As shown in figure 10-1, models for business process integration, business logic, and user interface play a key role on different levels of abstraction in the different development phases.

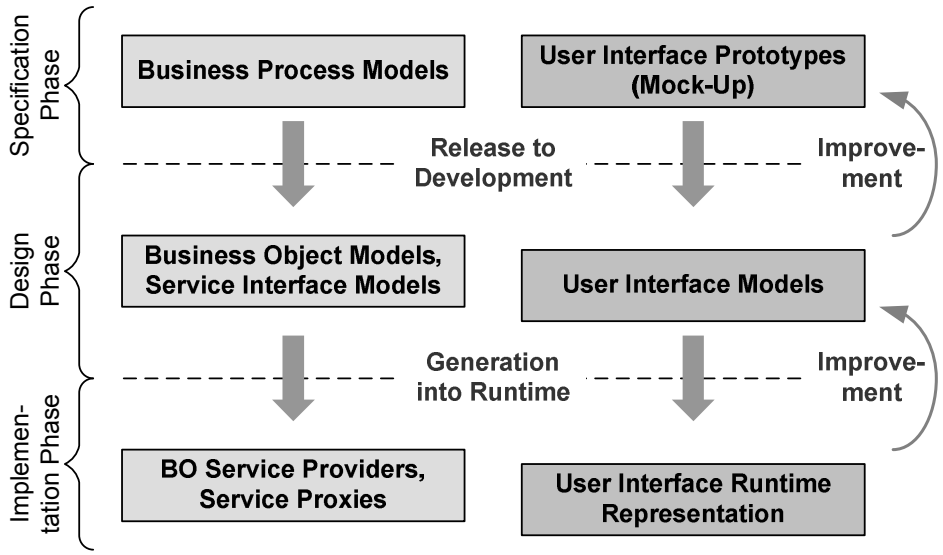


Figure 10-1 Modeling in the Enterprise SOA Development Process

During the specification phase, business requirements are represented as business process models comprising process flow models and process integration models, giving a static view of process components, their internal structure, and their relationships. The focus is not on specific business process functionality, but rather on integration and service orchestration between the process components.

During the design phase, details of the business application are modeled based on the process integration models defined in the specification phase. The business object model describing structure, behavior, and core service interfaces is created as well as the service definition for compound service interfaces. In addition, the UI prototypes created in the specification phase are now worked out in detail as user interface models.

In the implementation phase, actual program code is created. Code skeletons of service provider classes, proxy classes, interfaces, and data types, among other entities, are generated from Enterprise Services Repository (ES Repository) definitions. The results serve as the foundation for subsequent manual programming. In most cases, the user interface is implemented without the need for any coding. Based on ES Repository definitions the UI prototypes created in the design phase are refined in SAP NetWeaver Visual Composer.

All models created in AP / ByDesign are stored in ES Repository. The tight linkage between models, definitions, and runtime entities

ensures that no breach of consistency occurs between any of the steps.

As a result of the model-driven development approach, developers and customers gain a much better and quicker understanding of how business processes are built into the software. This eases the development and implementation process, allows rapid adaptation according to changing business needs, and can result in lower TCO.

In the following sections we describe the phases of the model-driven development process in detail, using the example integration scenario SELL FROM STOCK introduced in chapter 3.1

10.2 Specification

The development process begins with the specification phase, which maps business requirements to appropriate models. The specification phase starts with portfolio planning; selecting the business requirements and the corresponding functions and features that should be provided by the software application. The result is a list of business scenarios together with a description of the functions and features to be implemented.

As the next step, we translate the results of portfolio planning into models, as shown in figure 10-2. This requires the following activities.

- Define the business processes on a non-technical level
  Result: Process flow model
- Identify the functional building blocks (business objects, process components, and deployment units) that are required by the selected integration scenario
  Result: Business object map
- Define the interaction between the process components of the integration scenario
  Result: Integration scenario model
- Define the communication between process components in detail, including the identification of required process agents and compound services as well as the modeling of the message choreography
  Result: Process component model and process component interaction model

In parallel, the required application user interface (work centers and portal applications) are identified and screen elements, including field visibility, navigation, and buttons, are defined. This results in a set of initial UI mock-ups.

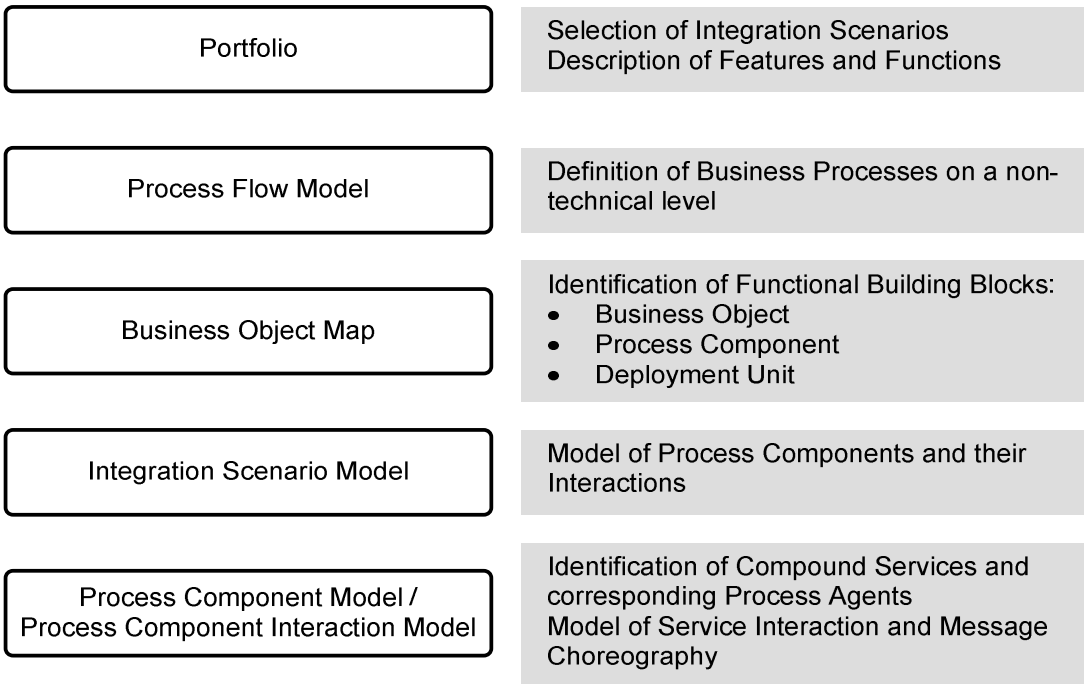


Figure 10-2 Specification on Different Levels of Abstraction

10.2.1 Process Flow Model

The model designer defines the business processes using process flow models, that follow the standardized Business Process Modeling Notation (BPMN). Process flow models are used on business level.

A process flow model shows activities, flows, and gateways that control the flow (decisions, fork, merge).

The example in figure 10-3 shows the process flow of sales order processing with material.

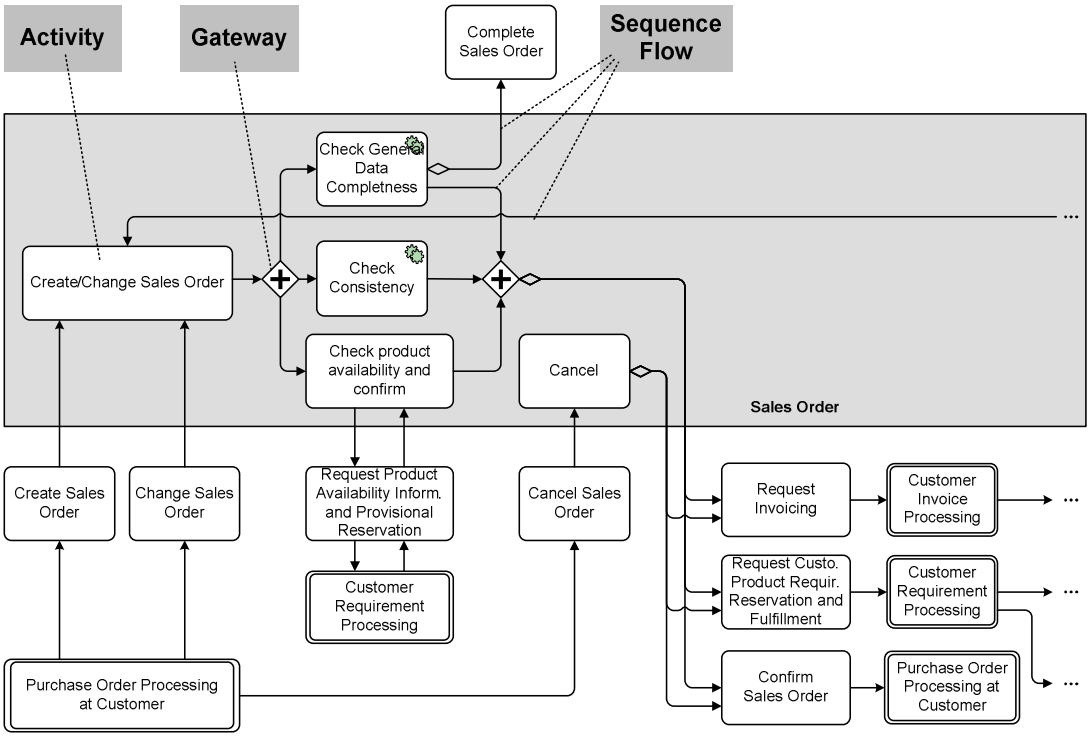


Figure 10-3 Process Flow Model of Sales Order Processing with Material (Excerpt)

10.2.2 Business Object Map

To create the business object map, the model designer should identify functional building blocks in the following order, starting with the smallest:

- Business objects
- Process components
- Deployment units

The business object is the capsule for data and core services. Business objects that are closely related semantically are combined within a process component. Process components that should not be distributed on different systems are included in one deployment unit. Figure 10-4 depicts the graphical representation of the functional building blocks.

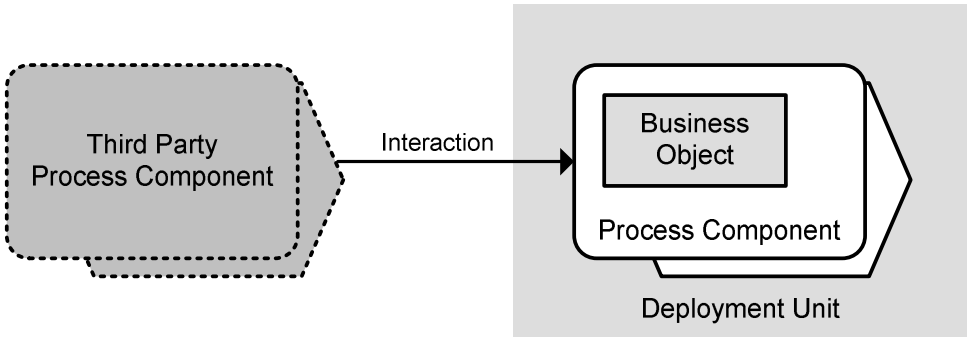


Figure 10-4 Functional Building Blocks in Business Object Maps

For the integration scenario SELL FROM STOCK, we identify four business objects – SALES ORDER, CUSTOMER REQUIREMENT, CUSTOMER INVOICE REQUEST, and CUSTOMER INVOICE – located in three process components (see figure 10-5). We assume that the purchase order is sent from another company's purchasing application. Therefore, we will not model and implement it. Because we want to run each process component on a single system, we distribute the process components on three deployment units. The resulting business object map is shown in figure 10-5).

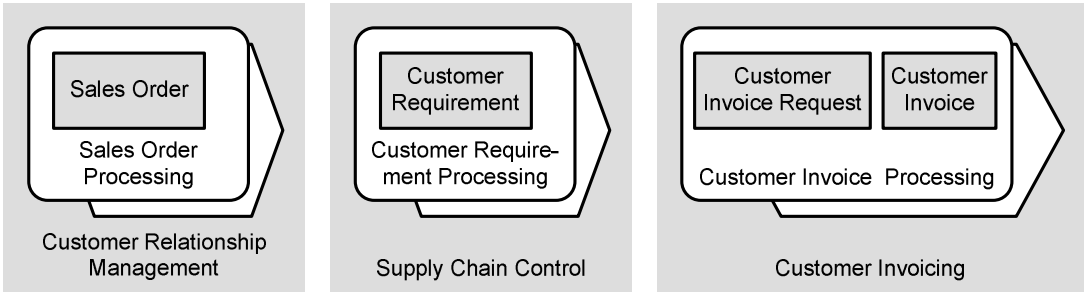


Figure 10-5 Business Object Map for Integration Scenario "Sell from Stock"

The next step is to model the interactions between the process components in an integration scenario model.

Business object maps as well as integration scenario models and the more detailed related models are created using the ARIS™ for SAP NetWeaver toolset integrated in ES Repository.

10.2.3 Integration Scenario Model

An integration scenario model, as depicted in figure 10-6, provides the highest level of abstraction of the actual design and implementation. It is based on three entities:

- Deployment unit
- Process component
- Interaction between process components

A process component realizes certain business process steps. Typically, it represents a well-defined functional area of an enterprise. But process components are not bound to a single integration scenario. They are reusable modeling entities that can be part of different integration scenarios. The interaction between two process components is represented by an arrow indicating the process flow (see figure 10-6).

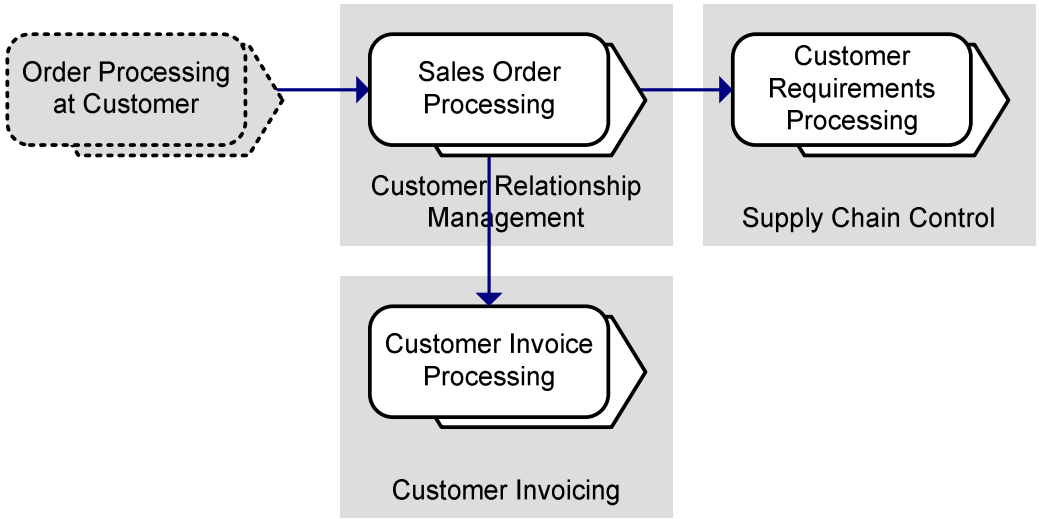


Figure 10-6 Integration Scenario Model "Sell from Stock"

The integration scenario model is the entry point to the complete set of related detailed models. Within ES Repository, it is possible to navigate through the models: clicking on an entity shows the detailed model. The process component model describes the details of a process component. For cross-deployment unit and business-to-business communication, interaction is refined further in the process component interaction model.

10.2.4 Process Component Model and Process Component Interaction Model

To get a more detailed picture of the interaction between process components, we first need to identify the required compound service interfaces and process agents. We use them within the process component model and process component interaction model to define the concrete interaction between services and the message choreography.

The process component model and the process component interaction model are built from the same entities. In fact, the two models represent two different views on the same content. All elements of the process component model as well as the process component interaction model correspond to runtime entities.

Business objects and their services are in the center of a process component model. The model specifies the compound services that business objects provide. On the other hand, the process component interaction model presents a connection-oriented view. It contains only those parts of two related process components that deal with the interaction between these process components. It includes message types and message choreography.

The model designer has to create a process component model manually. For convenience, a draft version of the process component interaction model is created automatically from the process component model.

10.2.4.1 Process Component Model

The process component model provides details of the internal structure of a process component. It describes the business objects, their inbound and outbound service interfaces together with their operations, as well as the corresponding inbound and outbound process agents.

The example in figure 10-7 outlines the entities used in the process component model of the process component SALES ORDER PROCESSING.

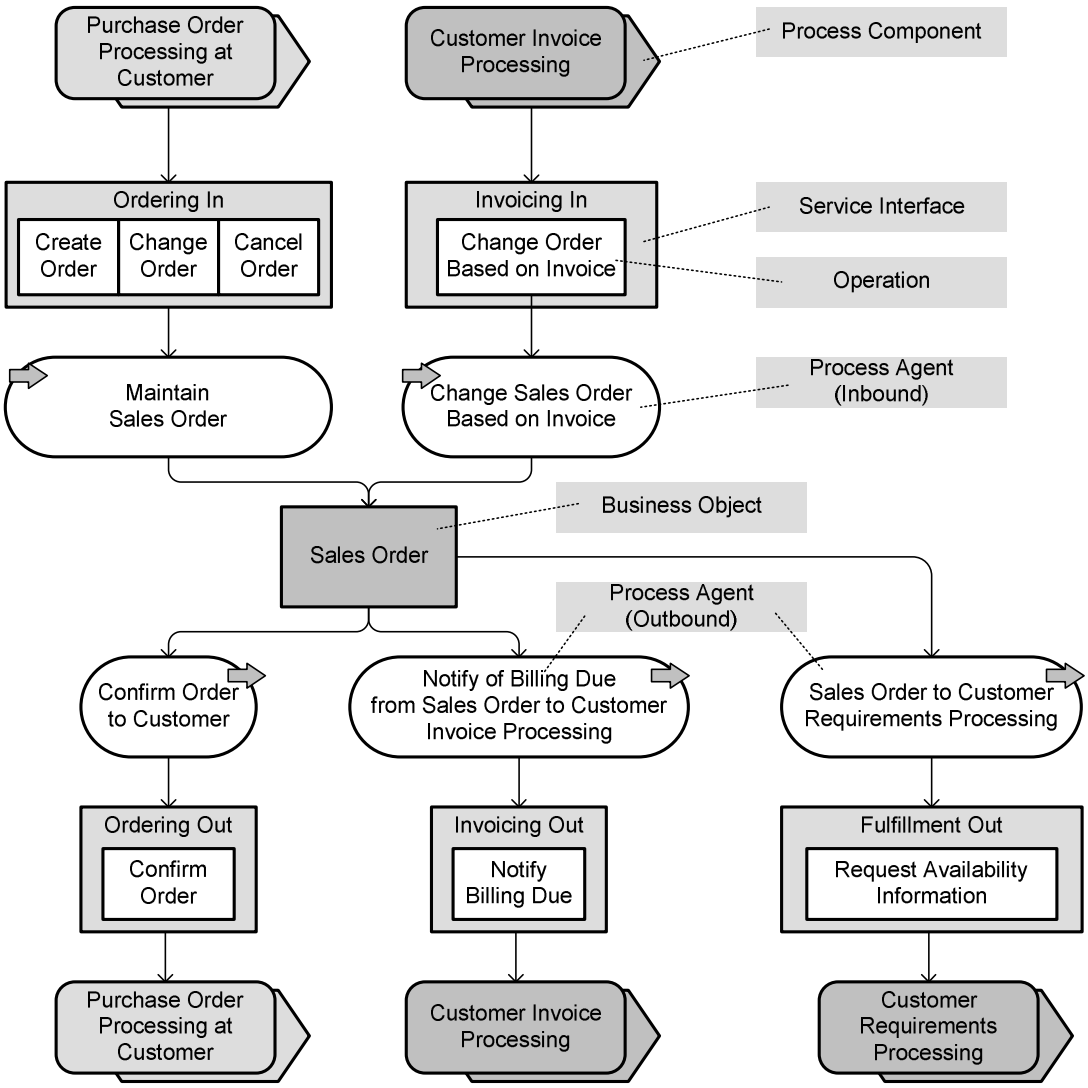


Figure 10-7 Process Component Model for Sales Order Processing

10.2.4.2 Process Component Interaction Model

The process component interaction model defines the interaction and message choreography between two process components. It describes participating business objects, inbound and outbound process agents, service interfaces and operations, and the messages involved.

For our example, we use the interaction between the process components SALES ORDER PROCESSING and CUSTOMER INVOICE PROCESSING (see figure 10-8). The process component interaction model displays the following entities:

- Business object
- Process component
- Inbound and outbound process agent

- Service interface
- Operation
- Message data type

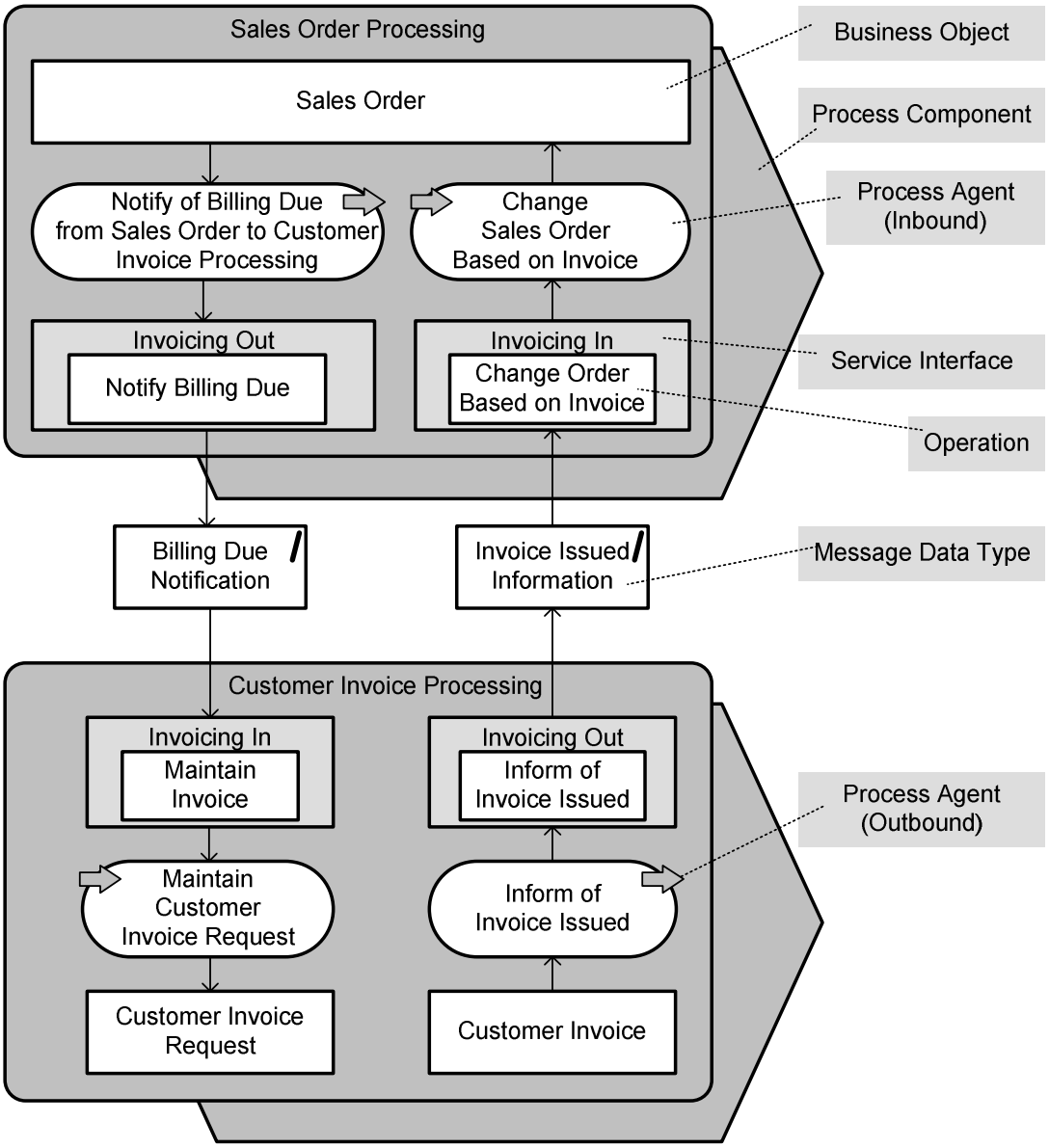


Figure 10-8 Process Component Interaction Model from "Sell from Stock"

The example in figure 10-8 shows asynchronous communication between the process components SALES ORDER PROCESSING and CUSTOMER INVOICE PROCESSING.

The business object SALES ORDER is the initiator of the communication. One outbound process agent – NOTIFY OF BILLING DUE FROM SALES ORDER TO CUSTOMER INVOICE PROCESSING – is registered for this business object. Whenever the process agent needs to trigger

a communication, it uses the operation NOTIFY BILLING DUE of the compound service interface INVOICING OUT. In this case, a message of message data type BILLING DUE NOTIFICATION is sent to the operation MAINTAIN INVOICE of the compound service interface INVOICING IN, which subsequently starts the inbound process agent MAINTAIN CUSTOMER INVOICE REQUEST. This process agent triggers create, update, or delete operations of CUSTOMER INVOICE REQUEST instances as required.

In the case where the INVOICE REQUEST leads to a CUSTOMER INVOICE, the communication goes the other way via the outbound process agent INFORM OF INVOICE ISSUED and the operation INFORM OF INVOICE ISSUED of the service interface INVOICING OUT. The corresponding message data type is INVOICE ISSUED INFORMATION. At the process component SALES ORDER PROCESSING, the message is processed by the operation CHANGE ORDER BASED ON INVOICE of interface INVOICING IN and the inbound process agent CHANGE SALES ORDER BASED ON INVOICE.

The process component model and process component interaction model form the connection between process integration modeling and design. All entities specified in these models, namely business objects, process agents, service interfaces and operations, have an equivalent on the design level. Both models are used to generate design entities such as the business object model. Furthermore, the models are used for development planning, documentation, testing, business configuration, and rollout. For example, the central configuration system (see chapter 8.1) uses the process component interaction model for activating process agents and configuring the message choreography required for running a specific business process variant, as described in the following section.

10.2.5 Business Process Variant Type Models

The concept of process components allows to realize various possible business processes that use the business objects and service interfaces of the process component. We refer to these possible business processes as business process variants.

In AP / ByDesign, similar business process variants are classified by types, the business process variant types (BPVTs), which provide properties to control business processes as follows:

- BPVTs control the interaction between process components. During configuration, the selection of a BPVT results in the activation of certain process agents, as shown in figure 10-9, where BPVT1 activates the 1st and 4th process agent and BPVT2 activates the first 3.
- When a process agent is triggered, it checks whether the next action is relevant for the current BPVT. This check is part of the condition evaluation as described in chapter 5.3.1.
- Business objects may provide attributes specific for a BPVT. Furthermore, business object instances store information about the BPVT in which they were involved.

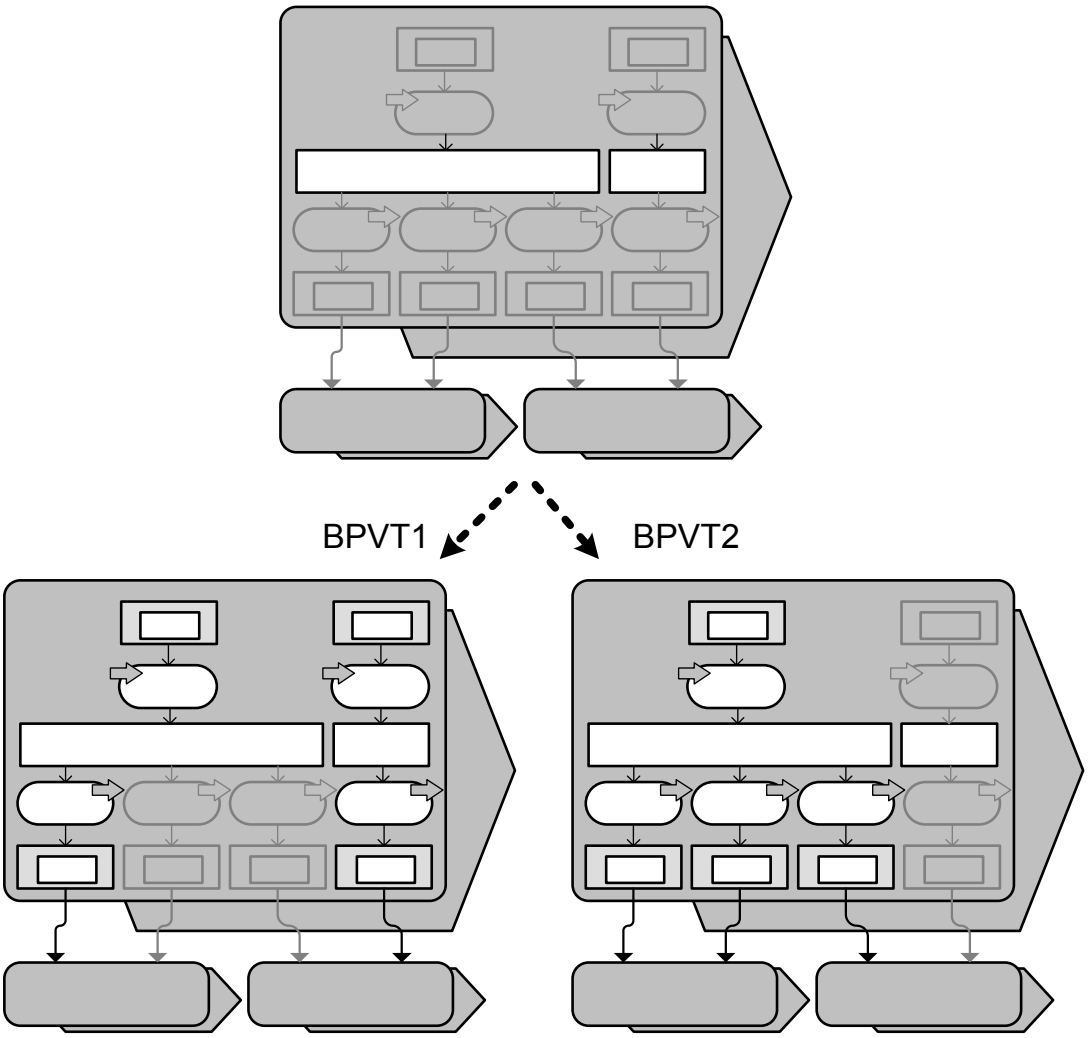


Figure 10-9 BPVTs Determine Active Process Agents

A BPVT is assigned to exactly one process component. Since it controls the process agents and provides further configuration, each process component must have at least one BPVT.

During configuration, BPVTs are linked, among other business content, to the business options of the business adaptation catalog, which is described in chapter 8.1.

10.2.6 Modeling UI Prototypes

Modeling user interface prototypes should start already during process integration modeling. The required roles, work centers, and floor plans, as well as the layout (color, fonts, and so on) of a screen can be defined independently of the business object model. When composing prototypes of actual screens, you need information about fields and values provided by a business object to present them on the user interface. Therefore, the design of the business object structure and the specification of the required fields on the screen have to be done in close cooperation.

UI prototypes should give a real picture of the target user interface and should be able to simulate user interaction. Potential end users can test the prototypes and check whether the screens fit to their needs. In the design phase, developers use UI prototypes as blueprints for the model of the final user interface.

10.3 Design

The design phase starts with the business object model, which is the foundation for the subsequent design of compound service interfaces and user interfaces. The following entities are worked out in detail in ES Repository:

- Business object model:
  Structure of nodes, including their attributes with their global data types. Core service interfaces are derived automatically from these structures.
- Service definition:
  Compound service interfaces with operations are designed using global data types.

- Process agent model[7]

Design entities provide the foundation for generating code skeletons for the corresponding runtime objects (see chapter 4.2). This results in a direct correlation between modeling and implementation entities.

ES Repository metadata, especially the business object model, is imported into SAP NetWeaver Visual Composer for modeling user interfaces. Therefore the design of business objects needs to be finalized before the user interface can be modeled.

10.3.1 Design of Business Objects

A business object model needs to be defined for each business object belonging to a process component. This includes the following (see figure 10-10):

- Definition of the business object's nodes and their structure with all associations
- Definition of the business object's data types containing all attributes of all nodes; each attribute is typed by a global data type
- Status and action model
- Definition of specific queries and actions of the business object

Business objects have to be defined carefully. Because their attributes are the basis for compound service interfaces and fields on the user interface, it's more difficult to change them later on. Therefore, some measures are required to ensure the quality of those definitions. SAP, for example, has introduced an internal process for the definition and approval of business objects used in all SAP applications.

During design, business objects are created from the process component model in ES Repository. Figure 10-10 shows an ES Repository screen shot of the design of the example business object SALES ORDER. We see the business object node structure on the left and the properties of the root node in the center. Additional tabs offer designing actions, associations, queries, and the status

[7] In the current release, ES Repository does not support detailed modeling of process agents.

and action model. It is also possible to view the WSDL description of the business object, which is generated from the definition.

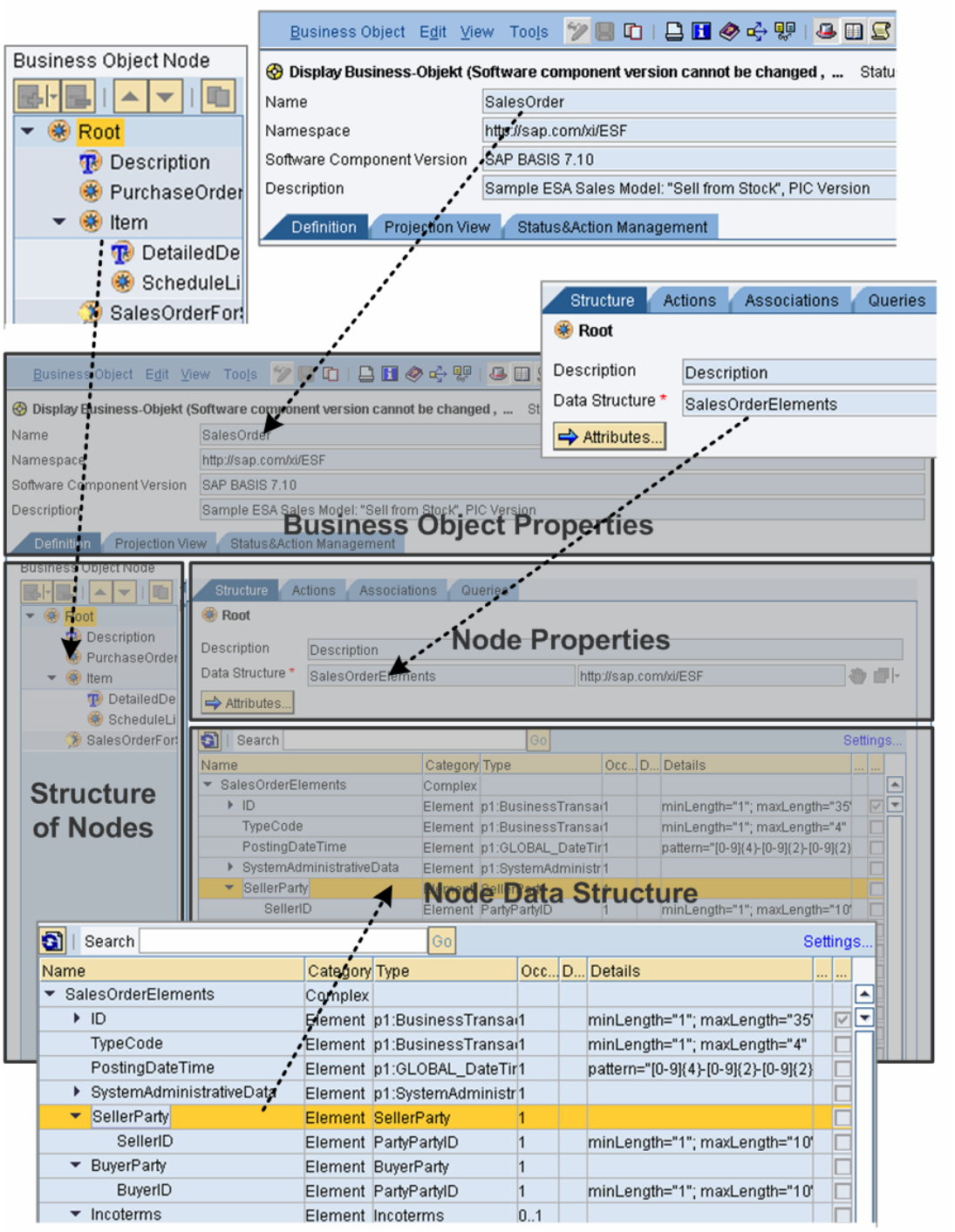


Figure 10-10 Business Object Model of Sales Order (Screen Shot from ES Repository)

Many business objects share common data. To minimize data overlap and to maximize the reuse of business objects, business

object templates and dependent business objects have been introduced to simplify the process of designing business objects.

10.3.1.1 Business Object Template

Similar business objects (for example CUSTOMER, SUPPLIER, and EMPLOYEE) share common attributes but also have private ones. To minimize the design and implementation effort, developers can use a template that includes all the common and private attributes of a subset of business objects.

To derive a specific business object from the template, the developer selects attributes needed for a new business object. The developer makes selections by a successive restriction of the available attributes by deactivating either entire business object nodes or single attributes in business object nodes. In this way, the business objects CUSTOMER, SUPPLIER, and EMPLOYEE are derived from the business object template BUSINESS PARTNER. Business object templates cannot be instantiated and are marked as templates in the ES Repository.

10.3.1.2 Dependent Business Object

Some generic business objects have high potential for reuse, but from a business point of view, they cannot exist independently from other business objects. To enable the design of generic business objects, we introduced the dependent business object. It is linked to a business object via a “dependent object” association.

A typical example is the dependent business object ADDRESS, which is embedded in many business objects but cannot exist independently.

10.3.2 Design of Compound Service Interfaces

For each compound service interface specified in the process component model, developers have to create a corresponding service definition during design time. First, they define message data types based on global data types, which include all parameters of the operation. Then they generate design entities of the service interfaces with their operations from the process component model. For each operation, they have to assign the corresponding message data types.

Developers define the following properties on the service interface level:

- Inbound or outbound interface
- Stateless or stateful service

For each operation, one can define whether synchronous or asynchronous communication should be used.

The screen shot in figure 10-11 shows the example service interface SALES ORDER PROCESSING INVOICING OUT with its operations NOTIFY BILLING DUE and NOTIFY BILLING DUE RECONCILIATION. It is a stateless outbound interface for triggering billing. The operation NOTIFY BILLING DUE is used to send the corresponding request asynchronously to the business object CUSTOMER INVOICE REQUEST of process component CUSTOMER INVOICE PROCESSING (see also the process component interaction model in figure 10-8). The assigned message data type is BILLING DUE NOTIFICATION.

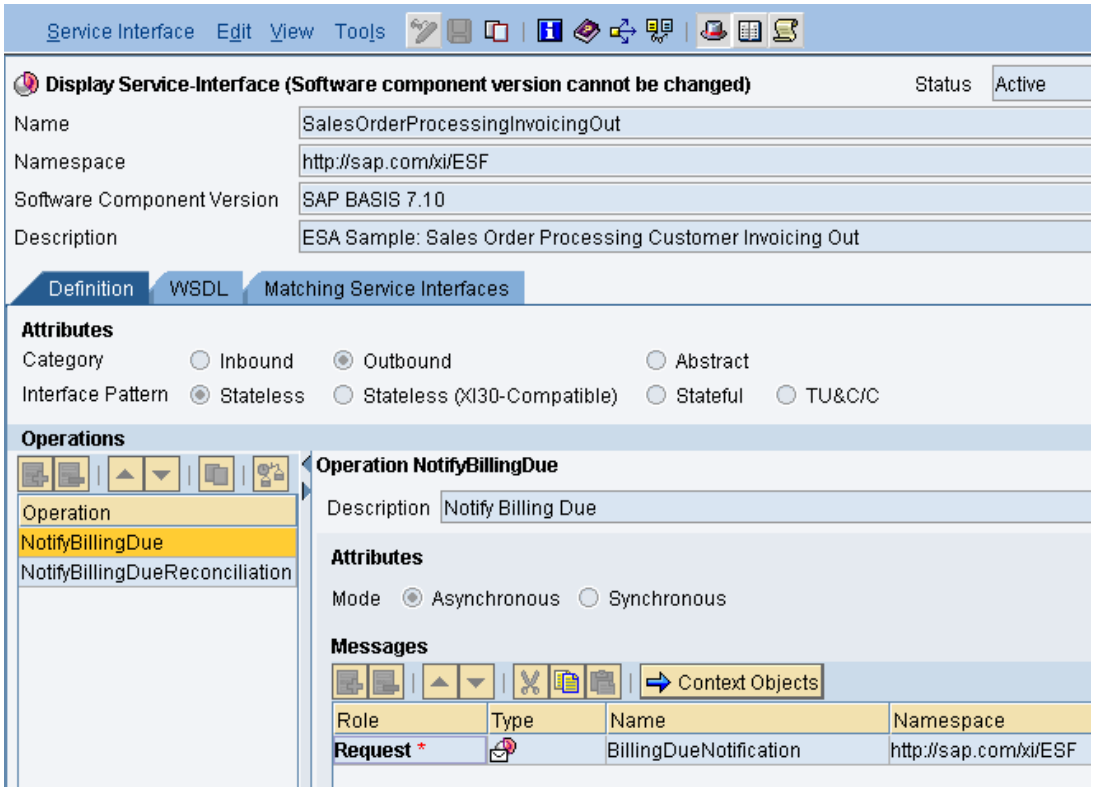


Figure 10-11 Service Definition of Compound Service Interface (Screen Shot from ES Repository)

10.3.3 Modeling the User Interface

Developers can create the AP / ByDesign user interface without writing code. To do this, developers have to perform three tasks:

- Define the layout via the portal stylesheet
- Create roles and the corresponding control and work centers in the portal content directory
- Model the user interface interaction and structure with SAP NetWeaver Visual Composer

In this section, we concentrate on the last activity, as the others are typical tasks to customize SAP NetWeaver Portal and are not specific for AP / ByDesign.

With SAP NetWeaver Visual Composer, developers model user interfaces by selecting and composing UI building blocks, which are bound to business object models imported from ES Repository. As explained in chapter 6, the UI building blocks (floor plan, UI pattern, and UI element) are implemented with Web Dynpro.

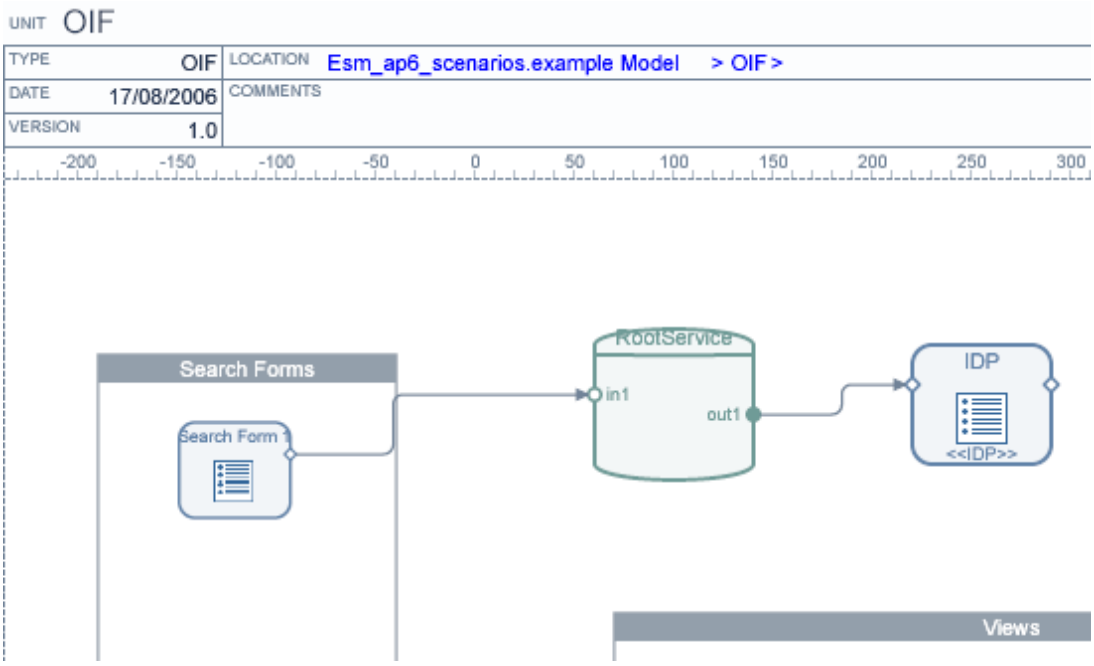


Figure 10-12 Draft user interface Model of Object Instance Floor Plan (Screen Shot from SAP NetWeaver Visual Composer)

The most important entities for user interface modeling are:

- INTERACTOR represents a piece of the user interface interacting which the user. Typical interactors are based on UI patterns such as SEARCH FORM, SIMPLE OBJECT DATA PATTERN, and IDENFICATION PATTERN (IDP).
- SERVICE represents data processing in the back end. It is either a single core service or a set of them.
- PORT serves as an entry and exit point of an interactor or service (in-port and out-port).

Each entity can be configured within SAP NetWeaver Visual Composer.

The modeling of a user interface starts with the selection of a FLOOR PLAN. Starting with the selected floor plan, the developer first creates a draft model and then refines it (see figure 10-12).

Figure 10-13 shows the final UI model for editing sales orders provided for the example scenario SELL FROM STOCK. Two in-ports – OPEN and BUSINESS PARTNER – define how a user can navigate to the screen. Either the user can navigate to the user interface directly from the work center – via the in-port OPEN, or he can select a business partner and navigate from that user interface to the business partner's sales orders for editing – via the in-port BUSINESS PARTNER. On the screen there are three search forms corresponding to the queries of the business object SALES ORDER, which we designed in the ES Repository. Each search form is presented as an interactor.

The in-ports and the search forms are connected to the service ROOT SERVICE, which provides access to QUERY and RETRIEVE core services of the business object. The search result is displayed using an identification pattern represented by the corresponding interactor IDP. The IDP provides a YOU CAN ALSO AREA (interactor YCA) and four views: one is the list of search results; the others display the details of one selected sales order instance (OVERVIEW, STATUS, and ITEMS).

The final user interface is shown in figure 10-14. The user can select the views of the IDP in the contextual navigation panel. Below that, the user has access to the search forms and the YOU CAN ALSO section.

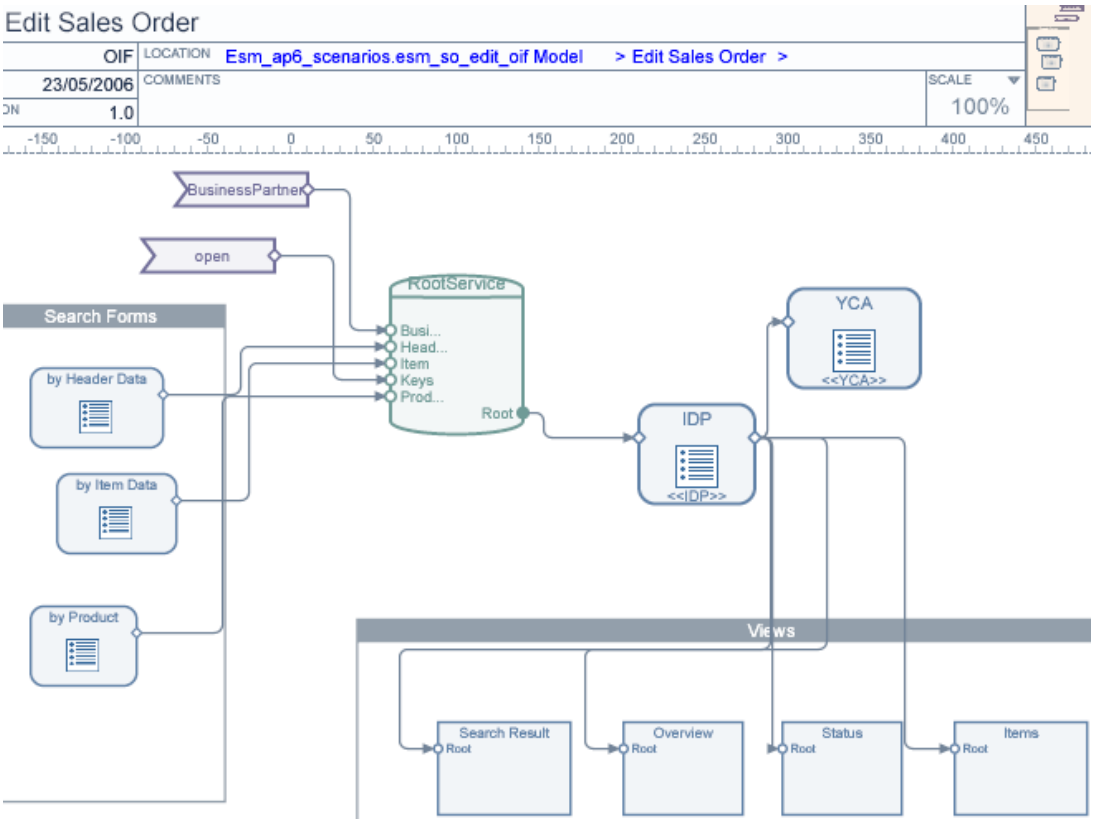


Figure 10-13 UI Model for Editing Sales Orders (Screen Shot from SAP NetWeaver Visual Composer)

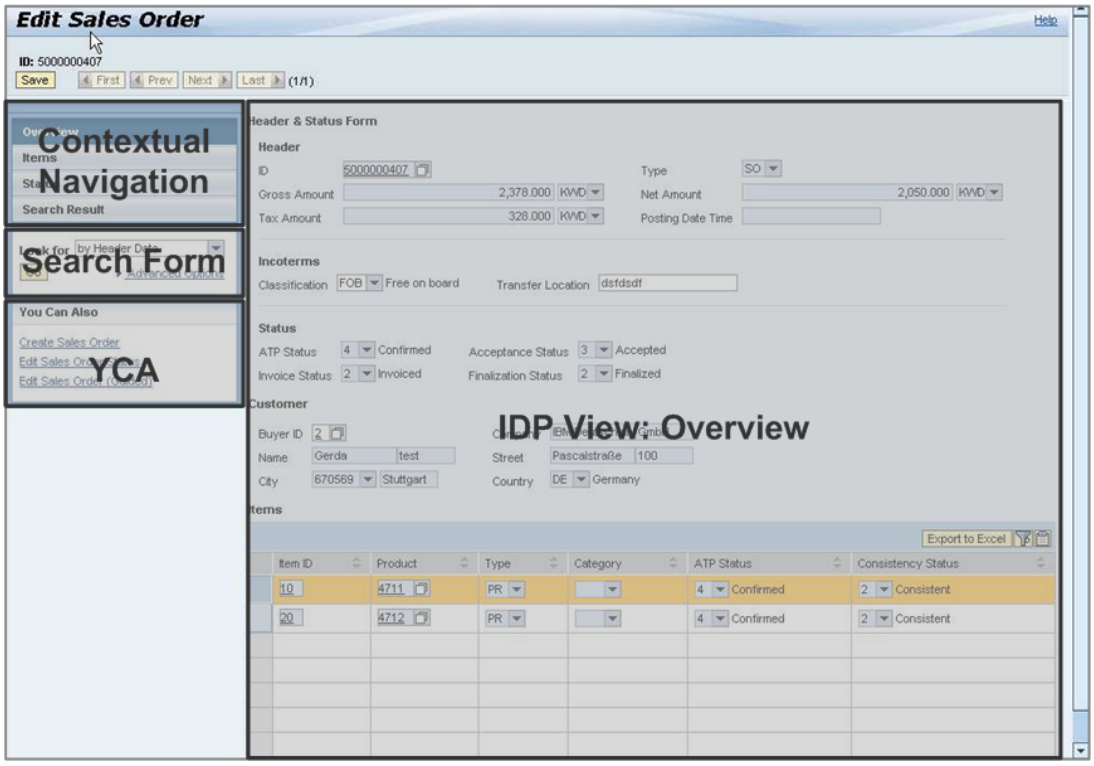


Figure 10-14 Final user interface of Object Instance Floor Plan for Editing Sales Orders

10.4 Implementation

In the implementation phase, executable code is created. Based on the programming language-independent ES Repository definitions, skeleton ABAP™ classes are generated in the development environment. These generated classes have to be enriched by the appropriate business and integration logic.

Developers connect their development environment to ES Repository, browse the content, select model entities, and generate skeleton classes[8] for them (see figure 10-15). Skeleton class generation is supported for the following:

- Business objects
- Compound service interfaces
- Process agents

Consult chapter 4 for details of service provider classes for business objects and proxy classes for compound services. Process agents are described in chapter 5.

In contrast to the generation of provider classes, the implementation of the user interface does not require writing additional code. The runtime entities of the user interface can be completely generated from the SAP NetWeaver Visual Composer user interface models.

[8] In the SAP Business ByDesign development environment, generated skeleton classes are also called “proxy” for historical reasons.

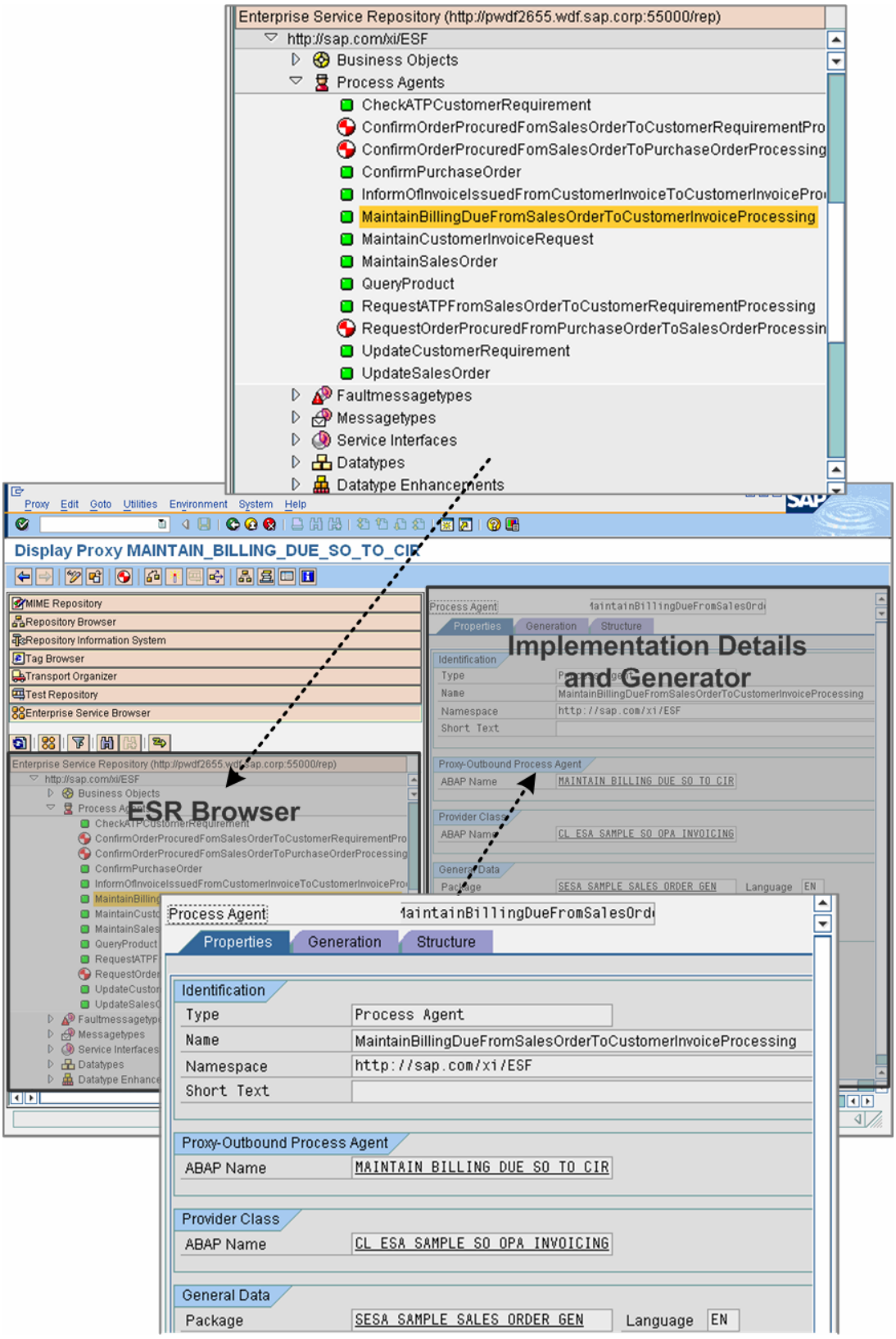


Figure 10-15 Skeleton Class Generation and Maintenance in the Development Environment

10.5 In a Nutshell

Model-driven development ensures consistency between the models describing the system and the implementation.

In the specification phase, a set of abstract and coarse-grained models are created and refined, beginning with a portfolio of business requirements and features and ending with models describing process components and their interaction. UI prototypes complement these specification models.

During design, architects and developers create more detailed technical models, starting with the process component models of the specification phase. Design includes business object modeling, service definition, and process agent modeling. All design models are stored in ES Repository, which plays a key role in model-driven development in AP / ByDesign. Based on the UI prototypes, the user interface is modeled as well.

The majority of implementation entities, such as classes, data structures, and user interfaces, are generated from design models. In some cases, the generated code has to be enriched with business logic or process logic. In other cases, such as the user interface, additional coding is usually not required.

The AP / ByDesign architecture, together with the support of powerful tools for modeling and code generation, help architects and developers to focus on business logic and user interaction.

11 Summary and Outlook

Conversations about business software still tend to center on individual products for functional areas, such as enterprise resource planning (ERP), customer relationship management (CRM), supply chain management (SCM), and so on. Why do we need all these different product categories? And why aren't they consolidated into a single unified solution? The authors believe that, in the long term, companies will no longer think about ERP, CRM, and SCM as separate entities. Instead, they are already today more interested in productivity plus flexibility and innovation. And they are looking for software product architectures that address this demand.

For midsize companies, AP / ByDesign, as derived from the enterprise SOA blueprint, is SAP's answer to this need: a rich, service-oriented set of business functionality and technologies, preintegrated, easy to use, easy to support, and simple to extend and to innovate.

In this book, we focused on the technical architectural concepts behind AP / ByDesign and the underlying application platform. At the center, we saw the business process platform supporting the idea of collaborating service providers and service consumers, with services being standardized, reusable runtime resources performing specific tasks upon request by diverse service consumers. All these services are based on Web standards, designed and executed by the enterprise service infrastructure.

Services can be used by various service consumers like user interfaces, composite applications, and other business objects for different purposes. Loose coupling between service providers and service consumers provides the foundation for process flexibility, quick innovations, and increased software development productivity.

Service providers and service consumers can be located in one deployment unit interacting locally via core service calls, or in different deployment units with message-based communication between compound services governed by the process agent framework.

A model-driven approach for the entire software life cycle has been introduced to enable enterprises to achieve the full benefits from the new service-oriented architecture. With this, developers, partners, and customers quickly gain a much better understanding of how business objects, services, and processes are built into the software. This improves the development and installation process, helps to identify and utilize reuse components, and allows rapid adaptation according to changing business needs, thereby resulting in predictable TCO.

With enterprise SOA, SAP created the blueprint for bringing together service-oriented architecture and business applications. AP / ByDesign is SAP's first offering for midsize companies that builds on this blueprint. In parallel, the SAP Business Suite development is making huge progress in providing enterprise SOA to large enterprises as well. Although AP / ByDesign and the SAP Business Suite are different in the details of their technology, they follow the same enterprise SOA blueprint that supports a common modeling approach, common service definitions, and a common composition environment for composite applications.

Bibliography

[ABH04] Ali Arsanjani, Bernhard Borges and Kerrie Holley:
Service-Oriented Architecture.
DM Direct Newsletter, November 12, 2004

[BB03] Ruediger Buck-Emden, Jochen Boeder:
Integrated Business Applications – Service-Oriented Architecture is the Next Megatrend.
in: Informatik Spektrum, Volume 26, No. 05, October 2003 (in German)

[BBF+05] Norbert Bieberstein, Sanjay Bose, Marc Fiammante, Keith Jones, and Rawn Shah:
Service-Oriented Architecture Compass.
IBM Press 2005

[Erl05] Thomas Erl:
Service-Oriented Architecture – Concepts, Technology, and Design.
Prentice Hall 2005

[Gart05] Gartner Inc:
Gartner Says Midsize Enterprises Should Take a More Strategic Approach With IT Spending to Better Align With Their Business Needs.
Press release, Stamford, April 6, 2005

[KBS05] Dirk Krafzig, Karl Banke, Dirk Slama:
Enterprise SOA – Service-Oriented Architecture Best Practices.
Prentice Hall 2005

[KGT06] Andreas Knoepfel, Bernhard Groene, Peter Tabeling:
Fundamental Modeling Concepts.
Wiley 2006

[NFN+06] Valentin Nicolescu, Burkhardt Funk, Peter Niemeyer et al:
Developer's Handbook SAP Exchange Infrastructure.
SAP Press, 2006 (in German)

[Osw06] Gerhard Oswald:
SAP Service and Support.
SAP Press, 2006

[SAP07] SAP AG:
Basic Concepts of AP/A1S Architecture.
White paper, January 2007

[WoMa06] Dan Woods, Thomas Mattern:
Enterprise SOA.
O'Reilly, 2006

[xuPA06] XU Product Architecture (ed.):
ESA Architecture Series 2006.
SAP internal reports, February 2006

Appendix

A Notation of the Models

A.1 Business Models

As a result of the model-driven development approach, the majority of business processes, data and logic can be defined by modeling, as shown in chapter 10.

A.1.1 Process Flow Models

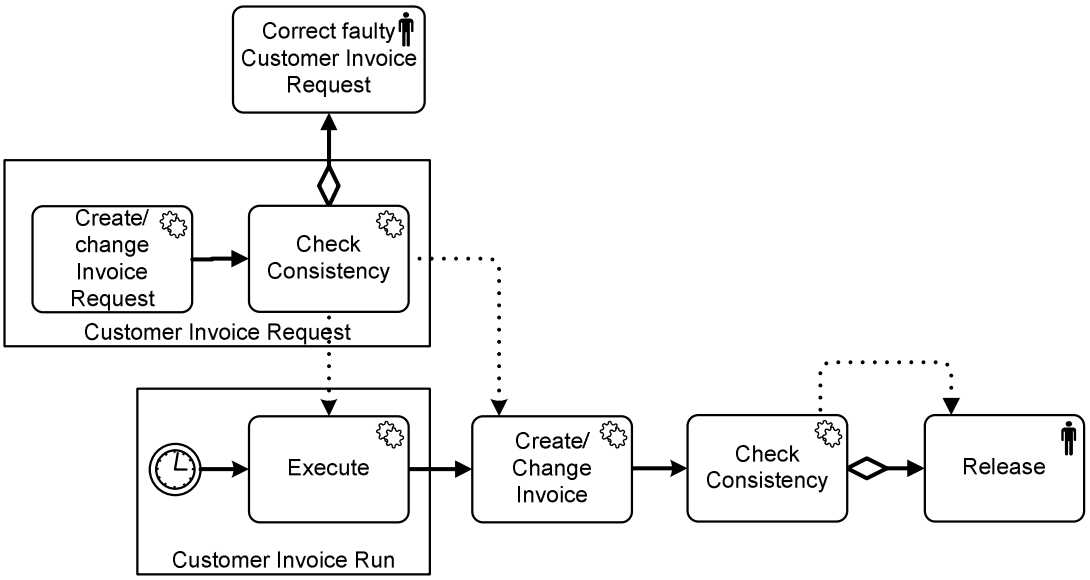
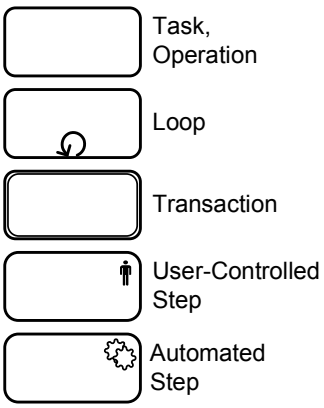
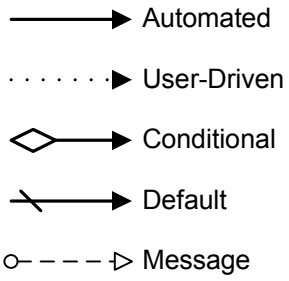
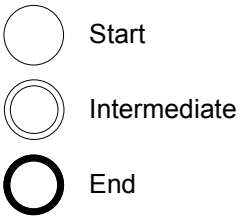


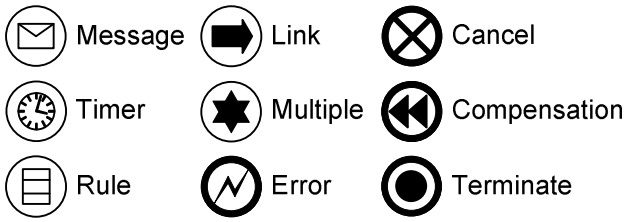
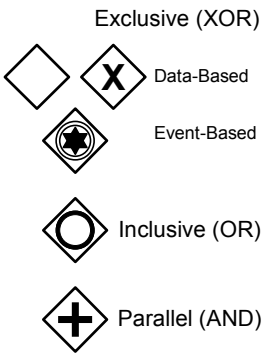
Figure A-1 Example of a Process Flow Model (Customer Invoice Processing)

A.1.1.1 Purpose

Process flow models show behavior on business level. They use an enhanced version of the standardized business process modeling notation (BPMN).

A.1.1.2 Diagram Elements

Activities  	An activity is a generic term for work that company performs. An activity can be atomic or non-atomic (compound). A task is an atomic activity that is included within a process. A task is used when the work in the process is not broken down to a finer level of process model detail. Loop: Tasks and sub-processes are repeated. A transaction is a sub-process that is supported by special protocol that insures that all parties involved have complete agreement that the activity should be completed or cancelled.
Flow  	A sequence flow is used to show the order that activities will be performed in a process. Sequence flow can have condition expressions that are evaluated at runtime to determine whether or not the flow will be used. For data-based exclusive decisions or inclusive decisions, one type of flow is the default condition flow. A message flow is used to show the flow of messages between two participants that are prepared to send and receive them.
Events  	An event is something that “happens” during thecourse of a business process. These events affect the flow of the process and usually have a cause (trigger) or an impact (result). Events are circles with open centers to allow internal markers to differentiate different triggers or results. There are three types of events, based on when they affect the flow: start, intermediate, and end. Intermediate events occur between a start event and an end event. It will affect the flow of the process, but will not start or (directly) terminate the process.

Event Type Dimensions	
Gateways 	A gateway is used to control the divergence and convergence of sequence flow. Thus, it will determine branching, forking, merging, and joining of paths. Internal markers will indicate the type of behavior control. • XOR – exclusive decision and merging. Both data-based and event-based. Data-based can be shown with or without the “X” marker. • OR – inclusive decision and merging • AND – forking and joining. Each type of control affects both the incoming and outgoing flow.
Pool 	A pool represents a participant in a process. It is also acts as a “swimlane” and a graphical container for partitioning a set of activities from other pools, usually in the context of B2B situations.

A.1.2 Process Integration Diagrams

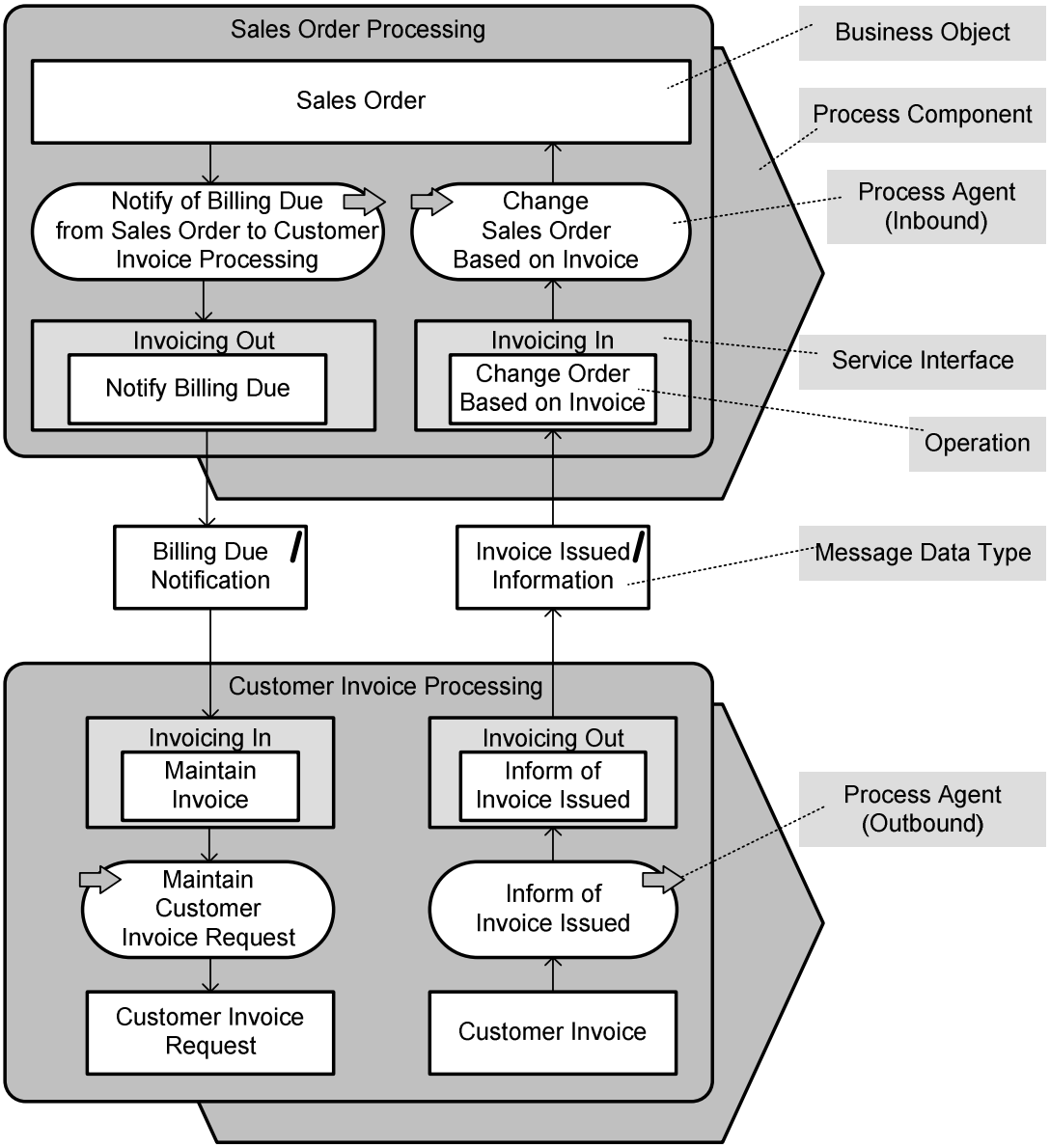
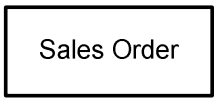
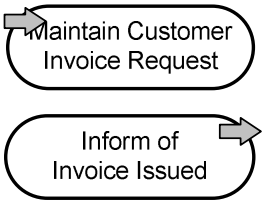
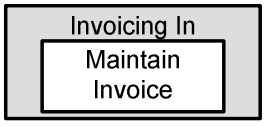
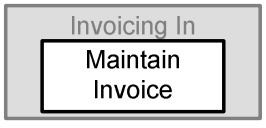
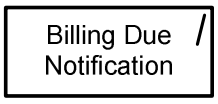


Figure A-2 Notation of Process Integration Diagrams

A.1.2.1 Purpose

Process integration models comprise integration scenario models, process component models, process component interaction models, and business object maps. They describe the internal structure of process components and their interaction, and use the same set of diagram elements.

A.1.2.2 Diagram Elements

Process Component 	Reusable building block for modeling integration scenarios. Process components bundle certain coherent business functionality and comprise one or several business objects. Example is CUSTOMER INVOICE PROCESSING.
Business Object 	Encapsulates business data and business logic and communicates via process agents. Examples are SALES ORDER or PURCHASE ORDER.
Inbound and Outbound Process Agent 	Process agent that assembles and processes messages sent or received via compound service interfaces. Synchronous and asynchronous message exchange is possible. Process agents can be separated in inbound and outbound process agents.
Compound Service Interface 	Web service that provides specific operations and parameters to access one or more business objects. Compound services use core services for accessing business object nodes.
Operation 	Smallest, separately-callable function of a compound service. It is described by a set of data types used as input, output, and fault parameters.
Message Data Type 	Data type of a message sent or received by a compound service. Used to show which messages are exchanged during process component interaction

A.1.3 Status and Action Diagram

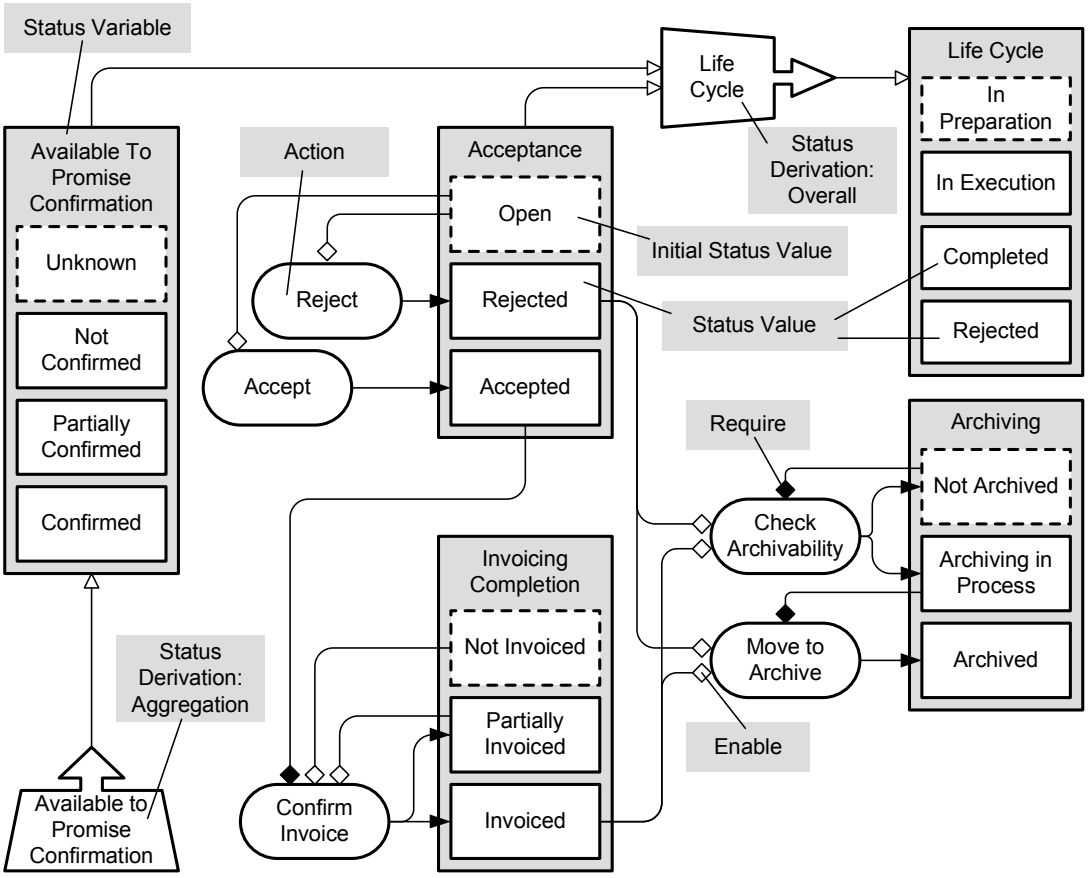


Figure A-3 Notation of Status and Action Models

A.1.3.1 Purpose

A status and action model describes the potential behavior of a business object node. It defines which actions are allowed to be performed if certain status values are reached. In the model constraints are visualized by the connections and arrows between status values and actions.

- Status variables and the corresponding list of possible values
- Actions that change status values
- Constraints that define how status values permit or inhibit the execution of an action

Status and action models are defined in the ES Repository. The modeling tools offer simulations to check the modeled behavior.

A.2 Technical Architecture Models

The architecture models shown in this book follow SAP's standardized technical architecture modeling (TAM), which defines a common language and a graphical notation for communication on model level.

TAM defines a UML 2.0 subset of diagram types with limited features that fits to SAP's needs for architecture descriptions. In addition, TAM extends the UML 2.0 meta model to incorporate FMC (www.f-m-c.org and [KGT06]) block diagrams as UML component / block diagrams.

The diagram types for structural descriptions are

- Component/block diagrams,
- Class diagrams, and
- Package diagrams (not used in this document).

Behavioral descriptions are covered by

- Use case diagrams (not used in this document),
- Activity diagrams,
- Sequence diagrams, and
- State machine diagrams (not used in this document).

Most of the diagram types are allowed to be used on two different abstraction levels: on *conceptual level* and *design level*.

Diagrams on conceptual level characteristically contain more abstract information, which describe architectural concepts in a broader context.

Diagrams on design level primarily focus on the reflection of code structures. Usually a model on design level exceeds the granularity on conceptual level. Therefore, the number of allowed and optional elements on design level is higher.

Since the majority of diagrams in this book show technical architecture on conceptual level, this section explains only those TAM diagram types and elements used in this book.

A.2.1 Component / Block Diagram

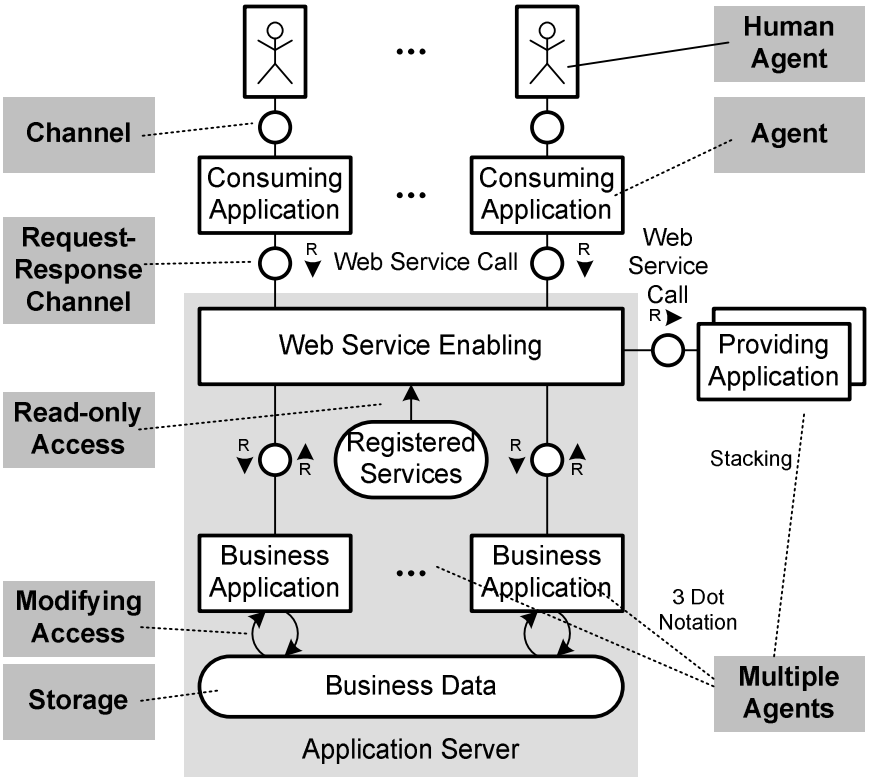


Figure A-4 Notation of Component/Block Diagram

A.2.1.1 Purpose

The component/block diagram is the most important diagram type to describe architecture on conceptual level. It is used to show the static structure of the system, and to provide a big picture view.

Agents perform operations; they read and write data in storages and communicate with other agents via channels. (Question: *Who does something?*)

Storages are passive; they only hold data that is accessed by agents. The arrows indicate the direction of data flow. (Q: *Where is the data?)*

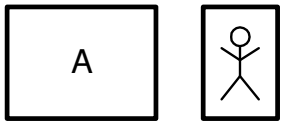
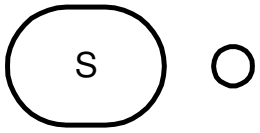
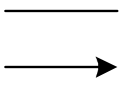
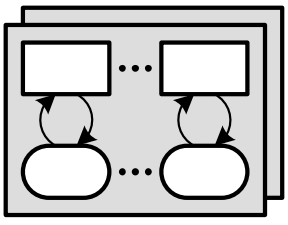
Channels are used by agents to communicate with each other. Arrows indicate the direction of data flow, the “R” indicates who initiates the request and then waits for the response. (Q: *Which agents communicate with each other? What data and which requests do they exchange?*)

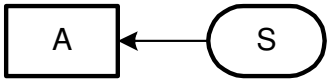
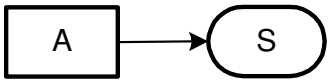
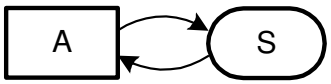
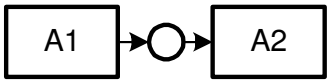
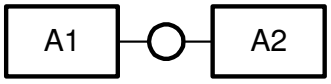
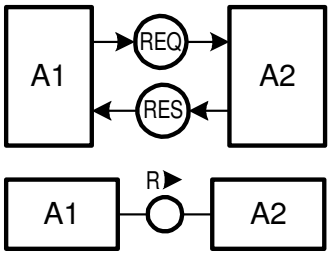
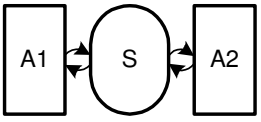
Block diagrams show a snapshot view of a system on instance level; to indicate multiple agents or storages, use three dots or stack the nodes.

Relation to other diagrams types: Use class diagrams to show how data relates that is located in storages or exchanged via channels. Use activity or sequence or state machine diagrams to express the behavior of agents and how they interact.

Component/block diagrams are based on FMC block diagrams. For more information about FMC, refer to www.f-m-c.org and [KGT06].

A.2.1.2 Diagram Elements

Agent, Human  Active system components	Serves a well-defined purpose and therefore has access to adjacent passive system components and only those may be connected to it. A human agent is an active system component that represents a human. (Note: nouns should be used for identifier "A")
Storage, Channel  Passive system components	A storage is used by agents to store data. A channel is used for communication purposes between at least two active system components. (Note: channels are usually depicted as smaller circles but may also vary like the graphical representation of storage places)
 Access type	Directed and undirected edges represent the kind of access an active system component has to a passive system component. The types of access are read access, write access and a combination of both. (Note: usually undirected edges depicting read/write access are used on channels whereas two directed edges also depicting read/write access are used on storages)
 Multiple Agents / Storages	Explicit visualization of multiple agents or storages by either staggering the elements or showing three dots between elements of the same kind.

 Read Access	Agent A has read access to storage S.
 Write Access	Agent A has write access to storage S. In case of writing all information stored in S is overwritten.
 Modifying Access	Agent A has modifying access to storage S. That means that some particular information of S can be changed.
 Unidirectional Communication Channel	Information can only be passed from agent A1 to agent A2.
 Bidirectional Communication Channel	Information can be exchanged in both directions (from agent A1 to agent A2 and vice versa).
 Request/Response Communication Channel	Agent A1 can request information from agent A2 which in turn responds (e.g. function calls or http request/responses). Because it is very common, the lower figure shows an abbreviation of the request/response channel.
 Shared Storage	Agent A1 and agent A2 can communicate via the shared storage S much like bidirectional communication channels.

A.2.2 Class Diagram

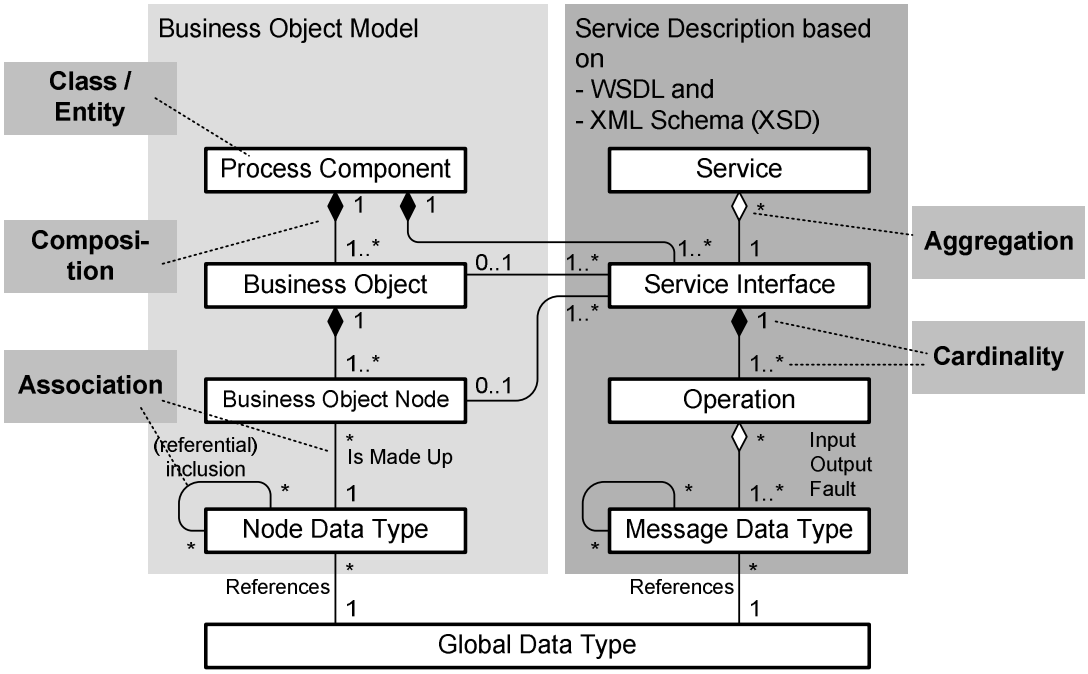


Figure A-5 Notation of Class Diagram

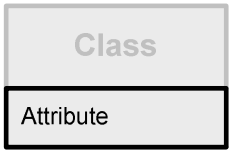
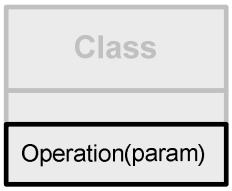
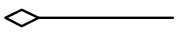
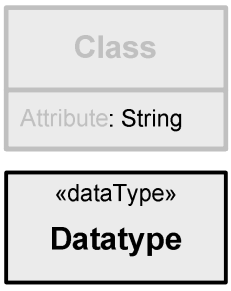
A.2.2.1 Purpose

A class diagram shows the system's entities and their relationships. It is used to explain terms and their relations on high level (glossary, topic map), or to show the structure of data to be processed.

Relation to other diagram types: Block diagrams show where data is stored or transferred in the system.

A.2.2.2 Diagram Elements

Classes **Class**	Entity types, such as service, document, or address

Attributes 	Classes can have attributes of certain types.
Operations (Methods) 	Operations that can be performed by a class.
Association 	Representation of relationships. Optionally, roles can be annotated at the association's ends.
Aggregation 	"has-a" relationship with weak linkage
Composition 	"has-a" relationship with strong linkage
Specialization 	Visualization of "is-a" relationships between classes.
Data types 	Preferably used for typing of attributes and operation parameters and return values. Alternatively, it can be visualized as a special classifier defining the data type.

A.2.3 Activity Diagram

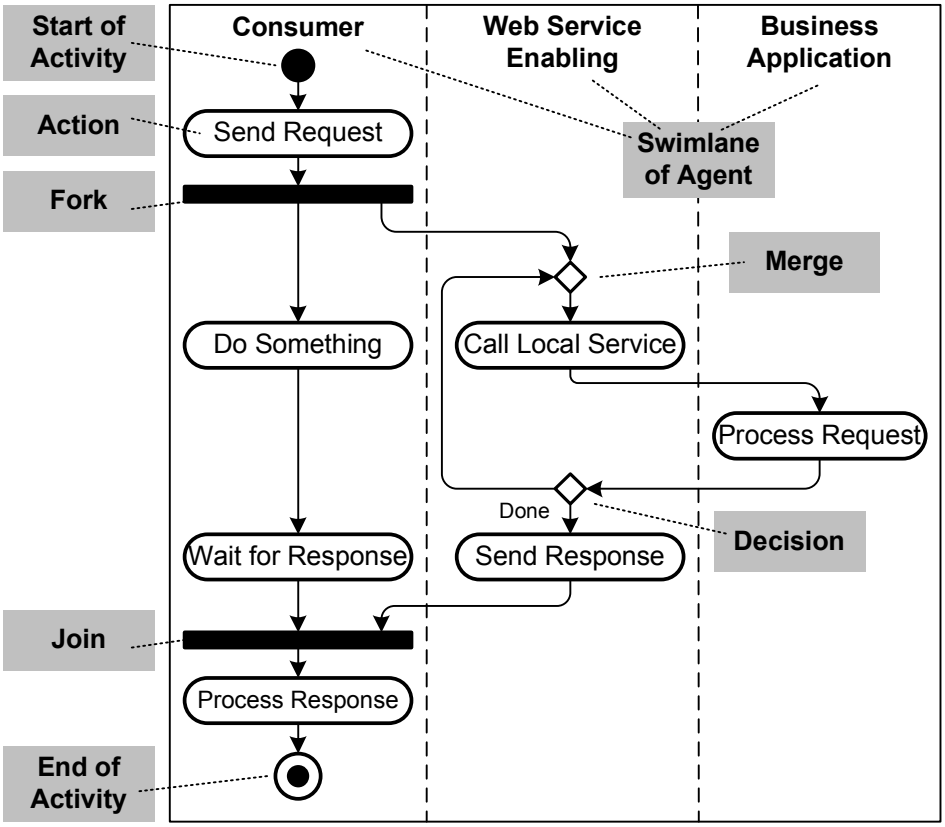


Figure A-6 Notation of Activity Diagram

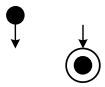
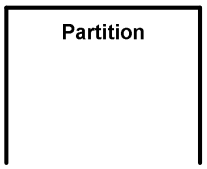
A.2.3.1 Purpose

Activity diagrams show the behavior of one or more agents of a system in greater detail. In contrast to the sequence diagram, you can show loops or concurrency in a more intuitive way. Using swimlanes, you can assign actions to specific agents.

Actions are utilized to show certain units of operation. *Control flow* (and optionally *object flow*) is visualized as connecting arcs between the actions. Optionally, activity diagrams can describe interactions of different entities by visualizing boundaries as swimlanes. Please notice that the whole sequence of actions (including *initial* and *activity-final node*) is called *activity*.

Relation to other diagram types: Block diagrams show the connection between the agents via channels, and access to storages.

A.2.3.2 Diagram Elements

Actions 	Steps in execution
Control Flow 	Connections between actions showing the order of execution and time dependencies
Initial Activity-Final Node 	Initialization and termination of an activity.
Decision / Merge 	Show basic mechanisms of the control flow (if, loop, …)
Fork/Join 	Parallel execution and synchronization points.
Swimlanes 	Partitioning the activity according to different contextual entities (e.g. objects, namespaces, and subsystems). Activities are unambiguously assigned to the particular entity they belong to.

A.2.4 Sequence Diagram

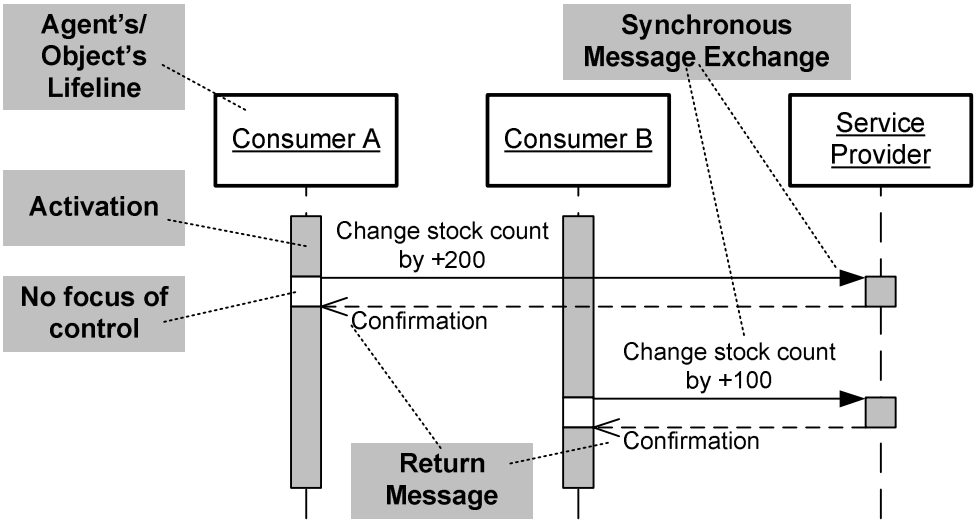


Figure A-7 Notation of Sequence Diagram

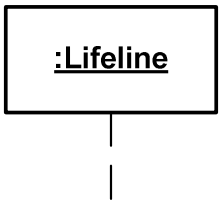
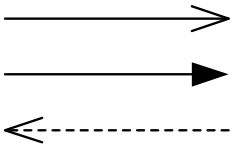
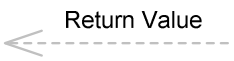
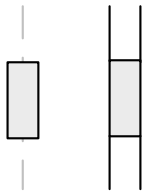
A.2.4.1 Purpose

Sequence diagrams show how agents interact and communicate in chronological order. They should be used on instance level, that is they should show one typical sequence.

The central aspect covered is message exchange between the objects. Although UML defines all elements necessary for complete behavioral descriptions, it is recommended to concentrate on sequential flows without many conditions or parallel execution. Characteristically, the resulting diagrams are more exemplary than generic.

Relation to other diagram types: Block diagrams show the connection between the agents via channels, and access to storages. To show more details of processing, use activity diagrams.

A.2.4.2 Diagram Elements

Lifelines 	All entities participating in a particular interaction are objects. Each object owns exactly one lifeline. On conceptual level, also human interaction can be shown.
Messages 	Objects exchange messages (asynchronous, synchronous, return message). This also includes the special case of messages originating and arriving at the same lifeline.
Return Values 	Synchronous calls usually require return values. On conceptual level, the return messages might be left out for reasons of simplicity.
Method Activation 	Visualization of activation on a lifeline. Additionally, differently shaded activation without focus of control is allowed to be shown if the object is (actively) waiting for an answer of a synchronous call.[9]

[9] The notation variant differentiating between focus of control and without control was part of the first versions of UML, starting with 0.8. It is helpful in situations, where detailed communication with all returning messages is shown, because it then emphases the main thread of control in the overall diagram. However, it can not be applied in all situations, therefore it is optional.

B Glossary

Term	Definition
Agent	Event-driven program entity that acts partially autonomously to accomplish certain tasks or to make decisions. AP / ByDesign differentiates process agent, business intelligence agent, and task agent
Analytical Engine	Part of SAP NetWeaver Business Intelligence that performs queries on info providers and delivers corresponding results
Application	Collection of software components that supports business processes required to address specific business needs; example is AP / ByDesign
Application Platform	Part of business process platform that provides core business functionality. It is structured into one foundation for generic functionality and multiple deployment units for business scenario-specific functionality.
Association	Named unidirectional semantic relationship between two business object nodes
Business Configuration	Adaptation of a business application to specific business needs by selecting certain options or fine-tuning predefined settings
Business Intelligence Accelerator	High-performance analytics appliance based on the SAP search engine and a specific hardware architecture using scalable blade servers
Business Intelligence Agent	Agent that extracts business object data from the OLTP data store for analytics
Business Object	Class of entities with significance to a business. A business object encapsulates business data and business logic and is accessible exclusively via core services. Examples are SALES ORDER or PURCHASE ORDER.

Business Object Node	Semantically related set of attributes of a business object. Different business object nodes are connected by associations.
Business Object Template	Abstract business object that defines features common to a set of business objects
Business Process	Set of logically related business activities that produce a specific business result
Business Process Platform	Entirety of the SAP application platform and SAP technology platform; supports creation, enhancement, and execution of business processes
Business Process Variant Type (BPVT)	Class of similar business processes that share a common configuration
Business Task Management	Infrastructure to enable end users or applications to create, receive, manage, and complete tasks
Communication Pattern	Pattern that describes meaningful message choreographies between sender and receiver; example is request confirmation pattern
Composite Application	Application that consumes services provided by other applications, but does not provide own services
Compound Service	Web service that provides specific operations and parameters to access one or more business objects. Compound services use core services for accessing business object nodes.
Control Center	Part of the user interface. Home area of a user. Provides the most important information about the user's daily work.
Core Service	Service that provides a generic operation on a business object node. Business objects are accessible exclusively by core services. Examples are CREATE, RETRIEVE, or DELETE.

Core Service Interface Pattern	Template for the definition of standardized core service interfaces; example is the access pattern that groups core service operations for accessing and modifying business object nodes
Data Source	SAP NetWeaver BI entity which defines the subsets of business objects attributes to be extracted together with mapping and the name of the responsible BI agent
Deployment Unit (DU)	Grouping of semantically related process components that need to run together on one system; examples are PURCHASING or CUSTOMER INVOICING
Enqueue	Logical lock to protect business object data from simultaneous access from multiple users.
Enterprise Service	Service that provides business functionality and is published by SAP in the SAP Enterprise Service Workplace. Enterprise Services are structured according to a harmonized enterprise model based on process components, business objects and global data types. They are well documented, guarantee quality and stability and are based on open Web standards.
Enterprise Service-Oriented Architecture (Enterprise SOA)	SAP's blueprint for a service-oriented architecture (SOA) emphazising on the specific requirements of business solutions.
Enterprise Services Infrastructure (ESI)	Service-enabling infrastructure of business process platform that provides design-time tools and runtime engines for creating, providing, and invoking services
Enterprise Services Repository (ES Repository)	Design-time environment where application developers model enterprise services, business objects, and business processes according to the enterprise SOA meta model. These meta-level models and definitions are used to generate platform-specific representations.

Fast Search Infrastructure (FSI)	Infrastructure for generic implementation and execution of query core services using the SAP search engine for response time optimization
Floor Plan	UI building block that defines the basic structure of an application UI. A floor plan combines multiple UI patterns and represents a generic user work scenario.
Global Data Type (GDT)	Data type defined by SAP that represents business-related content in conformance with widely used Web and business standards. All business objects and service interfaces share the same pool of global data types.
Guided Procedure	Design and runtime environment for user-oriented composite applications
Info Provider	Specific SAP NetWeaver Business Intelligence data objects for storing and presenting data in a way that is optimized for reporting; examples are info cube and hybrid provider
Integration Scenario	Description of the integration aspects between all functional areas touched by an end-to-end business process; examples are SELL FROM STOCK or MAKE TO ORDER
Intergration Scenario Model	Abstract description of the interaction between process components within or across deployment units
iView	The entirety of portal component (portlet) and its container
Logical Unit of Work (LUW)	Transaction on application level. All updates performed during a LUW are collected and committed in one short database transaction.
Mega-Tenancy	Concept for system operation and application hosting that combines the sharing of common data and resources with the separation of customer-specific data

Operation	Smallest, separately callable function of a service. It is described by a set of data types used as input, output, and fault parameters.
Pattern	Description of a standard solution to a common problem in software engineering and design; examples are user interface pattern and core service interface pattern
Process Agent	Agent that assembles and processes messages sent or received via compound service interfaces. Synchronous and asynchronous message exchange is possible. Process agents can be classified as inbound or outbound process agents.
Process Agent Controller	Part of the process agent framework that triggers process agents and ensures that the steps of a process agent are executed in the predefined order
Process Agent Framework	Runtime infrastructure for initiating asyn-chronous and synchronous communication across deployment units
Process Component	Reusable building block for modeling integration scenarios. Process components bundle certain coherent business functionality and comprise one or several business objects; example is CUSTOMER INVOICE PROCESSING
Process Component Model	Abstract description of the internal structure of a process component, which includes business objects, process agents, and compound service interfaces
Process Component Interaction Model	Abstract description of the internal structure of interacting process components, which includes business objects, process agents, compound service interfaces, and message data types
Proxy	Representative of a program component. A proxy provides an interface and forwards requests and responses to the component; example is compound service proxy

SAP NetWeaver Business Intelligence (SAP NetWeaver BI)	Data warehouse complemented by advanced analytical functionality allowing near real-time analytical reporting
SAP NetWeaver Composition Environment	Infrastructure for developing and running composite applications that consume enterprise services. It includes a lean version of the SAP Java EE 5 application server and model-driven development tools.
SAP NetWeaver Visual Composer	Graphical modeling framework for user interface design
Service	Callable runtime resource performing a coherent set of tasks upon request by a consumer. A service is bound to a given IT system instance providing it.
Service Consumer	Software entity that invokes a service
Service Provider	Software entity that offers a service
Service Provider Class	Program class representing a business object by implementing the core services for all business object nodes
Signature	Parameters of an operation
Task	Piece of work that has to be accomplished by a person. Tasks are initiated by the task agent.
Task Agent	Agent that initiates a task, determines the responsible business expert, and forwards the task to the business expert's to-do list
Transaction	Transition from one consistent state of a system or an application to another consistent state
Universal Work List (UWL)	List of tasks, alerts, and notifications that is presented in a user's control center
User Interface Building Block	Standardized, reusable element of the user interface such as floor plan, UI pattern, or UI element

User Interface Pattern (UI pattern)	UI building block that serves one dedicated user task; for example, searching for and identifying business object instances
User Interface Element (UI element)	Elementary part of an UI pattern such as buttons, tables, trees, fields, and labels
Web Service	Service implemented and designed according to common Web standards, especially Web Service Definition Language (WSDL)
Work Center	User interface for a specific work area such as purchase order management or sales order management. A work center consists of a set of views organizing and supporting a user's activities in this area.
Web Dynpro	Technology to design and provide user interfaces for business applications, serving Web browsers and other UI clients

C Authors

Dr. Ruediger Buck-Emden joined SAP in 1990, where he has since held different management positions in development, product management, and strategic planning, including assistant to the executive board (Prof. Dr. Hasso Plattner), Vice President of Development R/3*, and Vice President of CRM Architecture and Technology. Currently Dr. Buck-Emden holds the position of the Vice President Architecture Knowledge Transfer in SAP's Office of the CTO, with focus on service-oriented enterprise architecture. Before joining SAP, Ruediger Buck-Emden worked in the Business Systems development department of Nixdorf Computer AG as a system engineer and development manager for network-based business applications. Ruediger Buck-Emden studied industrial engineering and computer science before doing his doctorate on computer-based information systems at Braunschweig Technical University, Germany. He is visting lecturer at the Hamburg-Harburg Technical University and the Heilbronn University of Applied Science and the author of numerous IT specialist books and publications.

Jochen Boeder studied mathematics, history and computer science before becoming a consultant and developer on the area of e-business for the SAP subsidiaries e-SAP and SAP Portals. At SAP he worked in product management for mySAP CRM and in research and development. Currently he is project manager in the Architecture Knowledge Transfer team of SAP's Office of the CTO. Jochen Boeder has co-authored several text books and other publications covering software topics. As member of SAP's central Product Architecture Team Jochen Boeder has been closely involved in the development of enterprise SOA.

Dr. Bernhard Groene studied electrical engineering and then worked for five years at SAP on the development of infrastructure services and business applications. He then taught system modeling for six years at the Hasso Plattner Institute in Potsdam, Germany, and earned a PhD in software systems engineering. His special area is modeling complex systems with the purpose of knowledge transfer. He is coauthor of a book describing FMC, a methodology for modeling complex systems. In 2006, Bernhard Groene joined the Architecture Knowledge Transfer team of SAP's Office of the CTO as a specialist for knowledge transfer.

Martin Luenzmann studied mathematics and chemistry at university in Hamburg and Heidelberg. Still a student he joint SAP in 1992, where he worked as technical writer in various development departments. In 1998 he joined SAP Labs North America in Palo Alto, where he co-authored several Made Easy Guidebooks and participated as user interface designer in various research projects. Back to SAP he worked in knowledge transfer for mySAP CRM and in research and development in the area of master data distribution and master data development. Then he was responsible for knowledge transfer projects in SAP's Product Architecture Knowledge Transfer team.

Index

C

D

E

T

U

V

W

X