

Enterprise Services Architecture

ENTERPRISE SERVICES DESIGN GUIDE

THE BEST-RUN BUSINESSES RUN SAP™



CONTENTS

Enterprise Services Design Guide	4
Who Is This Guide For?	4
Goals	5
Enterprise Services Architecture	6
Solution Design Overview	7
Improving Solution Fit and Adaptability Through Enterprise Services	8
– How Enterprise Services Create a Better Solution Fit	9
– The Challenge for Solution Designers	10
The Enterprise Services Repository: The Services Deliverable	10
Simple Example	11
Foundations for Enterprise Services	11
Basic Concepts of Enterprise Services	11
– Interface	12
– Messages	12
– Loose Coupling	13
– Service Operations	14
– Service Consumers	14
– Business Service Levels	14
Functional Business Components	15
Business Drivers	16
– User Productivity and Ease of Use	16
– Next Business Practices and Process Innovation	17
– Business Automation and Process Efficiency	17
– Deployment Flexibility	17
Functional Categories of Enterprise Services	18
– Component Services	18
– Process Services	18
– Entity or Engine Services	19
– Utility Services	20
Business Drivers and Functional Categories	20

Discovery and Design of Enterprise Services: A Partitioned Approach	21
The Discovery, Design, and Documentation of Enterprise Services	21
The Three Levels of Enterprise Service Design	21
– Level 1: Designing a Single Enterprise Service	22
– Level 2: Designing a System of Related Enterprise Services	22
– Level 3: Service Enabling Many Different Enterprise Applications	22
Design Contexts and Indicators for Enterprise Services	23
Level 1: Discovery and Design of Single Enterprise Services	24
– Indicators for Enterprise Services	24
– Single Service Design Guidelines	26
Level 2: Designing a System of Related Services	27
– Business Transactions	28
– Analytics	28
– Master Data	29
Level 3: Service Enabling Enterprise Applications	29
– A Service-Enablement Process	30
– The Procedural Model for Service Enablement	30
– General Guidelines for Designing Services and Common Indicators	31
Service Enablement – Design Context: User Productivity and Centricity	31
– The User Productivity and Centricity Design Context	31
– Business Process Analysis and Improvement	33
– User Productivity and Centricity Indicators	33
– Service Definition	35
– Other Design Contexts Related to User Productivity	35
Service Enablement – Design Context:	
Next Business Practices and Process Innovation (Composite Applications)	36
Service Enablement – Design Context: Business Automation and Process Efficiency	36
Service Enablement – Design Context: Flexible Deployment	36
Creating Your Own Design Context and Indicators	37
Describing and Specifying Services	38

Enterprise Services Design Guide

The Enterprise Services Design Guide is a gentle but thorough introduction to the process of designing collections of useful enterprise services. This document has two main tasks. First, it serves to educate those involved in designing and implementing solutions about the nature of enterprise services and the role they should play in a solution. Second, it presents a detailed process for discovering, designing, and describing an enterprise service.

This guide is gentle because the context is explained, not in an elementary manner as in a “For Dummies™” book, but quickly, for sophisticated engineers and business analysts. The explanations provide enough detail to make everything quite clear so that this document can be useful to the largest possible audience.

This guide is thorough in its end-to-end, top-to-bottom explanation of how to design and precisely describe and document enterprise services. Because enterprise services represent a new dimension of design, differing in both user interactions and business processes, even people with a great deal of experience in the field will need a thorough guide such as this one to fully grasp the concepts involved.

Perhaps the best way to understand the process of designing an enterprise service is to keep the following ideas in mind:

- Enterprise services bring to life the potential value of the Enterprise Services Architecture blueprint.
- Enterprise service design is a business-driven process that derives ideas for services by starting with an understanding of the business scenarios and business processes.
- Business scenarios and business processes may have to be understood in greater depth than ever before in order to discover the enterprise services that can be used to automate them in a flexible manner.

- Enterprise services are not designed in isolation. They may be designed to work together in groups that support each other or as single services that meet a variety of different needs.
- Enterprise services must be designed for reuse. Enterprise services should be designed to meet the needs of as many different business processes as possible, from existing business processes to emerging processes.

In other words, designers of enterprise services ideally must see deeper into business processes and farther into the future. While this is of course difficult in some ways, it should not make one think that enterprise service design is an impossible task. When one starts to approach the task of designing solutions from an enterprise services perspective, plenty of great ideas for services appear immediately as low-hanging fruit.

Who Is This Guide For?

This guide will be of interest to anyone involved in the process of designing and implementing solutions that use information technology to help meet the needs of businesses and other organizations. Specifically, this guide meets the needs of the following sorts of people, whether they work for large vendors like SAP or for smaller organizations like systems integrators, independent software vendors, or corporate IT departments:

- Business analysts
- Managers
- Product designers
- Architects
- Developers

The traditional process of creating IT solutions involves understanding the business processes being automated and the way in which those business processes could be controlled by a user interaction. For the purposes of this document, we call this process solution design. With the rise of Enterprise Services Architecture, a new task has been added to solution design — that of designing enterprise services.

The process of discovery and design will be used in many contexts and will be adapted in each of those contexts. Here are ways in which we expect the discovery and design of enterprise services to take place:

- Corporate business analysts will create enterprise services to enable processes that are unique to their organization.
- Solution managers at SAP will add enterprise services to the mySAP™ Business Suite family of business solutions to address current and emerging customer requirements.
- Systems integrators will create enterprise services to help automate an emerging next practice at a customer site.
- Independent software vendors will add enterprise services to enable their functionality, to meet more customer requirements, and to be combined with both custom services and services offered by SAP.

The task of designing IT solutions differs greatly depending on the goals of an organization and the role it plays in building a solution. A business analyst at a company that uses IT solutions has different concerns from someone designing a solution at SAP. Product designers at an SAP partner and software architects and engineers may have other perspectives.

This guide serves the needs of all of these readers by explaining the process of designing enterprise services in detail so that each type of reader will be able to apply the process to his or her circumstances. The guide covers how to design a single service as well as systems of services. It also describes how to run a program that enables many different applications with services. A business analyst describing an enterprise service for a single company will find guidance in this document, and so will a solution manager describing an enterprise service for thousands of customers. A business analyst at a company that uses IT solutions may end up using a much more streamlined version of the design process explained in this document, while someone involved in solution management at SAP may use this process as the basis for one that is even richer and more ornate.

One of the most exciting aspects of enterprise services is that they are designed to be shared and are built around mechanisms to support sharing. When enterprise services are created, they reside in the “enterprise services repository,” so everyone involved in crafting a solution can see them. SAP has started the ball rolling by announcing the “enterprise services inventory,” a collection of basic enterprise services that it continues to expand with new services. As time passes, more and more services will be added by SAP and others, making the enterprise services repository indispensable for solution designers.

Goals

Before the arrival of Enterprise Services Architecture, solution designers were concerned primarily with inventing and describing business processes, data flows and transformations, and user interactions in enough detail to allow architects and engineers to implement a solution. Now, in addition to these tasks, solution designers must be able to decide what enterprise services must also be provided as part of the solution and how these services will work together to meet the needs of the relevant business processes. This document aims to begin the process of building a competency for discovering, designing, and describing enterprise services by providing enough information to answer the following questions:

- What is Enterprise Services Architecture and how does it create value?
- What is an enterprise service?
- How can enterprise services be categorized according to the value they provide?
- How do enterprise services fit into existing IT solutions?
- How can enterprise services be categorized according to the different functions they may serve?
- What are common design challenges for enterprise services?
- How can enterprise services be invented or discovered from a business perspective?
- How do groups of enterprise services work together?
- How should proposals for enterprise services be evaluated?
- How should enterprise services be described in complete detail?

Of course, true competency in creating enterprise services can be gained only through experience. This guide, however, should accelerate the learning process and help avoid obvious mistakes.

This paper is divided into the following sections:

- This introductory section lays out the context for enterprise services design, explains the goals of the document, and covers the basics of Enterprise Services Architecture.
- Section 2, “Solution Design Overview,” briefly describes the general area of solution design so that we can compare the process of designing enterprise services to traditional practices. Although there are similarities, the business-driven perspective is a key differentiator for designing enterprise services.
- Section 3, “Foundations of Enterprise Services,” covers the underpinnings of enterprise services, including Enterprise Services Architecture, the business drivers and functional categories of enterprise services, how enterprise services fit into existing enterprise software solutions, and common design challenges.
- With that foundation in place, everyone will be on the same page with a clear context; section 4, “Discovery and Design of Enterprise Services: A Partitioned Approach,” then describes the discovery and design methodology in a clear and unambiguous fashion.
- Section 5, “Describing and Specifying Services,” describes how services are documented.

Enterprise Services Architecture

The reason enterprise services are worth thinking about at all is because of the value that can be created for IT through Enterprise Services Architecture, SAP’s blueprint for how applications should be constructed using services as the fundamental building blocks. It is not this document’s job to provide a lengthy explanation of Enterprise Services Architecture. Much more information can be found on SAP® Developer Network (<http://sdn.sap.com>) and on SAP’s public Web site. However, in order to be thorough and to set the stage for the material to come, a simple explanation must be provided.

Enterprise Services Architecture is the architectural solution to the latest challenges facing IT in its quest to support business in ever more challenging times. Enterprise Services Architecture is a better way to build applications, making it possible to solve many new problems with existing solutions at the lowest possible cost. Enterprise Services Architecture is not just an SAP architecture. It applies to and is used by many third-party vendors and consultants, because it shows how to create business value from technical constructs like Web services.

Based on open standards, Enterprise Services Architecture enables companies to expand the scope of the problem that packaged software solutions can solve and make configuring, customizing, and extending solutions easier. Enterprise Services Architecture also simplifies creating new solutions, called composite applications, which are constructed by combining existing services to solve new problems. A composite application is an application on top of other applications or application components that makes use of data and functions provided as services by underlying application or application components and combines them into a coherent business scenario. Enterprise Services Architecture delivers flexibility to business and enables services to become the language of business that brings IT and business onto the same page.

One of the key differentiators that SAP brings to Enterprise Services Architecture, compared with other approaches to service-oriented architecture, is that SAP will not just deliver a container for services, like technology-focused companies do, but will deliver working services based on mySAP Business Suite. The first 500 services SAP defined this year are just the tip of the iceberg.

SAP is pursuing Enterprise Services Architecture because it opens the door to a new way of building solutions based on modeling and other productivity-enhancing techniques that will make everyone involved in creating and customizing SAP solutions more productive. As enterprise services roll out, developers and business analysts working for customers, systems integrators, SAP

partners, and the SAP engineering team will spend more time on making applications better and less time on technical concerns that do not add value. But for Enterprise Services Architecture to work, high-quality enterprise services must be designed and implemented.

Solution Design Overview

This guide uses the term **solution design** as a convenient way to refer to all of the different activities that go into designing and creating an IT solution. For our purposes, solution design can be thought of as encompassing the following steps:

- Identifying a problem, understanding the business processes related to it, and gathering requirements for a solution
- Inventing a new idea for a product, component, or feature to meet certain requirements
- Evaluating and improving the idea and related business processes with all interested parties inside and outside of the organization
- Revising the idea and the business processes to ease implementation and take advantage of existing components through discussions with engineers who will build the solution
- Describing the idea comprehensively and unambiguously so that it can be implemented and communicated to all interested parties
- Implementing the idea

Before the introduction of enterprise services, this design process played out through this familiar set of steps, usually starting with research into problem areas and points of pain that resulted in some sort of requirements document. Ideas for solutions were then created, evaluated, and selected. Next, the solutions were described through use cases, user interface mock-ups, process flowcharts, and descriptions of data flows and transformations. All of this would lead to the ultimate description of the solution in an engineering requirements document that would be used as an implementation guide.

This process of solution design is carried out at various levels of detail, depending on the problem at hand and the needs of the organization. A business analyst at a company that uses IT solutions may combine all these steps into one document that is reviewed by the users and the IT department and then implemented in a highly collaborative fashion, with lots of informal communication and adjustments between the business analysts, users, and engineers. At a large company like SAP, different groups of people may be involved with formal methodology at each stage in addition to many different governance and review processes.

The introduction of enterprise services does not essentially change this process, but adds a dimension that opens the door to a new way of building applications that provides profound business benefits. In addition to the use case, user interface, business process, and data flow descriptions, the solution designers must also decide what enterprise services should be built as part of a solution and how those services will work together to support the business processes. As we will explain in the next section, by providing enterprise services as part of their solutions, companies increase their power to quickly assemble and reassemble solutions using services from all of the different software systems they may have in their infrastructure. What happens is that, as more and more enterprise services become available, the process of building a solution becomes one of deciding how to use existing services to work together to solve new problems, not building each new solution from scratch.

The solution designer's job has become more interesting in the world of Enterprise Services Architecture. In the old world, solution design was about meeting the requirements that had been discovered. In the world of Enterprise Services Architecture, the job of enterprise service design is one that not only requires the solution designer to look at implementing reusable components that will be useful in solving problems in the current solution, but will also be useful in solving future problems.

With even just a few enterprise services available as building blocks, Enterprise Services Architecture comes into its own and allows support for business processes to be crafted in a flexible manner as well as helping to manage the increasing complexity of IT solutions. Here are a few ways that enterprise services achieve these goals:

- Enterprise services provide flexibility about how services are implemented. Because enterprise services are essentially a definition of an interface and the behavior that each operation should have, they can be implemented by a functional business component, an enterprise application, or custom development.
- Enterprise services allow segmentation of the problem area, which allows us to control complexity and the tightly coupled, brittle interaction between different pieces of the solution. Configuration influences the (functional) behavior of one single building block, whereas service orchestration (also a specific kind of configuration) reflects the (main) process flow of the overall end-to-end business process.
- To map the business reality to the IT solution, one can compare the requirements to the abilities of the offered solution for each functional business component separately, rather than for a whole product. In many cases, the offered components can be configured to meet the requirements to a large extent; in some cases a customer-specific component may be implemented. The higher granularity of the components guarantees a better way to cover the requirements, as shown in Figure 1 later in this guide.
- Enterprise services allow easy orchestration and reconfiguration simply by reordering service execution and enabling business rules to follow different paths through the process model based on a process template delivered with the “standard” solution.

It is easy to make the mistake of thinking that enterprise service design is a technical task that is similar to the process of designing application programming interfaces (APIs). Indeed, most examples of Web services that appear in trade publications and marketing materials perform highly technical functions that do not provide

much business value. Designing enterprise services is not technical, but is really an extension of the job of analyzing business processes. Instead of just describing the business process, the solution designers must now craft enterprise services that can support the business process. Enterprise services must represent a complete business function in order to function properly as the “language of business” that is the basis for the reuse and flexibility that Enterprise Services Architecture promises.

Improving Solution Fit and Adaptability Through Enterprise Services

This section puts meat on the bones of the idea of designing enterprise services through a business-driven process. We present a step-by-step process of analyzing a business and its IT needs along with an analysis of how enterprise services actually live up to the promise of Enterprise Services Architecture and enable better solutions to be constructed.

The business-driven approach described here is the one used by SAP to document its understanding of the needs of various industries. The SAP solution manager’s job is to understand industry and application areas from a business perspective to understand how SAP solutions should support processes with existing functionality and to discover new functionality that should be added. Solution managers use a tool called a solution map to record the key business scenarios, business processes, and process steps in an industry as a first step of demonstrating how SAP software will provide a solution. An understanding of the process of creating a solution map will be instructive for solution designers of all types, even though solution maps are SAP centric.

A solution map is a description of a broad area in which SAP is providing a solution, usually in an industry like high tech or chemicals. Inside this broad area are subareas, called scenario groups. (Specific solution maps for industries and other related maps of business processes, as well as a tool for composing solution maps, can be found at www.sap.com/solutions/businessmaps.)

For example, the quote-to-cash scenario group for the high-tech industry describes the challenges facing companies in accepting complex orders and getting them fulfilled.

Each scenario group can have several different *scenarios*.

In quote to cash, there are scenarios for quotation, contract, order management, and fulfillment.

Then, within each scenario, there are **business processes**. These are essentially descriptions of what the systems do to automate the scenario. In this example, a business process might be an available-to-promise process that allows someone taking an order to find out quickly what the company has the capacity to make.

Finally, you have a **process step**. One step in the available-to-promise process could be checking the inventory of required parts at a supplier.

There are several important things to notice about this structure. At the higher levels – the solution map, the scenario group, and even the scenario – the definitions concern business processes in an industry, not what a specific software system does. And nowhere in the hierarchy just described is an SAP product mentioned. The solution management process is about describing the business challenges at a high level and the processes that are involved in supporting them. This is the essence of the business-driven approach. Only after these scenarios, processes, and process steps are completely understood are they linked to SAP solutions in mySAP Business Suite, like the mySAP ERP solution and so forth.

How Enterprise Services Create a Better Solution Fit

Now, in the ideal case, the fit of the standard solution offered by SAP or other vendors – or from custom development to the business processes described in the solution map – will be perfect. A perfect fit means that all the needs of a company will be met by the standard solution after it is configured.

In the real world, a standard solution is a perfect fit only for business processes that have become well understood, standard commodities. Most of the time there is some sort of functionality gap between the standard solution and the user requirements. Configuration – that is, use of metadata to adapt the behavior of the standard solution – helps close some of that gap, but most of the time the gap still exists.

One way of closing that functionality gap would of course be to build a solution that meets a larger scope of needs, one that is more configurable to meet even more requirements. But this approach is limited by the fact that enterprise solutions succeed by solving common requirements across companies – and there are only so many common requirements. To close the functionality gap significantly, there must be an easier way to address uncommon, unique requirements.

Enterprise services allow a standard solution to fill more of the gap by creating reusable building blocks that can be easily combined to meet new requirements. In a sense, this could be thought of as configuration, but because the approach is radically different, it really goes beyond configuration into a new realm. Enterprise services help fill the functionality gap with the right-size building blocks, ones that are big enough to support business processes without being too small and making the job of recombining them too complex a task, as shown in Figure 1. It is crucial to get the scope of enterprise services right at design time, and much of the methodology described in this document is designed to make sure that the scope of each enterprise service is correct.

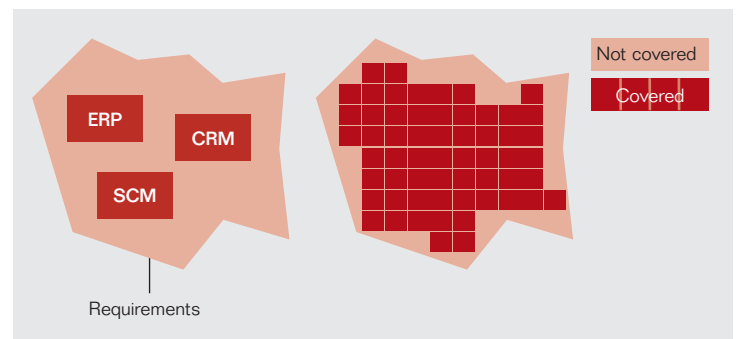


Figure 1: Solution Fit with Enterprise Services

The Challenge for Solution Designers

Now we can see the real challenge facing designers of enterprise services. It is from a deep understanding of the scenarios and business processes that enterprise services with the right scope will emerge, not from looking at the functionality of existing solutions. For example, the available-to-promise process might be a great candidate for a single service or perhaps for a collection of services, depending on how many of the component services could be useful in other processes. Solution managers must use their knowledge of business processes to make these trade-offs.

Another lesson for solution designers is that after you understand the service you need, you must connect them to what exists, a process that may require trade-offs to make implementation easier. At the later stages of the process, technical concerns influence the process of enterprise services design, as they do at some point in most design processes.

Enterprise services may be the most challenging design problem facing business analysts and solution designers. Not only must enterprise services be constructed to work together to serve the needs of a single business process or scenario, but solution designers must also anticipate how these services might be used in other business processes and scenarios. The number of factors and constraints in the process of designing enterprise services is generally larger and involves more business processes and scenarios than traditional solution management does.

This complexity and interrelatedness suggests that enterprise services will likely be designed in related groups of services that work together in addition to being used as isolated entities to encapsulate a system or important software component.

While this discussion of solution design should help clarify what exactly solution designers will be doing when designing enterprise services, it is worth pointing out that the key element in the process, as in most interesting design problems, is the quality of the invention of the design. The key challenge then for solution

designers is to become good at inventing services. The solution designer's ability to invent the right structure for services is what starts the process off. The rest of the methodology is there to make sure that mistakes are not present in the invention.

The Enterprise Services Repository: The Services Deliverable

Once enterprise services have been defined, they must be brought to the attention of solution designers, business analysts, architects, and others who will use them. Where do we keep the enterprise services? How do people find out about them? How do people use them?

The answer to these questions is the **enterprise services repository**, which can be thought of as the next evolutionary step in progression that began with the ABAP™ programming language business object repository and continued with the idea of the UDDI directory, which is part of Web services standards.

The enterprise services repository is the design-time repository of service objects for Enterprise Services Architecture. These objects include:

- Business objects
- Service interfaces
(with associated operations, message types, and data types)
- Process models
- Business scenario and business process objects
- Mapping objects

The enterprise services repository contains the description of the interface of the enterprise service, which is defined by a file in the Web Services Description Language (WSDL), a semantic description of the service, and a link to the business processes.

The enterprise services repository is open to input from many contributors. SAP will define enterprise services, and so will partners and customers. Once in the repository, all services will be available for reuse.

The **enterprise service builder** is the tool for creating and editing service objects, models, and interfaces contained in the enterprise services repository. It is an evolution of the SAP NetWeaver® Exchange Infrastructure (SAP NetWeaver XI) component's integration builder, as the enterprise services repository is an evolution of the SAP NetWeaver XI integration repository. An enterprise services repository browser allows people to examine which services are available in the repository.

Simple Example

The goal of this document is not only to drive home the idea of enterprise services in a memorable way, but also to provide readers with tools to explain the concept to others who will be involved in the process. To serve both of these purposes, we provide a simple example of enterprise services in a nontechnical context.

Imagine an analog watch. A watch has a face with hands on it that indicate the time and workings on the inside that keep the hands moving to show the correct time. For our purposes, we can think of the face of the watch as something analogous to the user interface of a software system, and the workings of the watch as similar to the programming code that does the work of the software system.

What then is an enterprise service in terms of this example? An enterprise service based on a watch might be defined as a service that used the workings of the watch to create a stopwatch service. The working of the watch could be used to create a service that had an operation for starting a timer, stopping a timer, displaying the current value of the timer, and resetting the timer. All of these operations could be grouped into a stopwatch service that could be displayed on a different user interface. In this way, an enterprise service for our watch reuses the basic workings for a new purpose.

Foundations for Enterprise Services

Now that we understand how enterprise services expand the world of solution design, we will turn to a more thorough description of what enterprise services are and how they work to create value.

In this section, we will cover:

- Basic concepts of enterprise services
- Functional business components
- The business drivers of Enterprise Services Architecture
- Functional categories of enterprise services

Basic Concepts of Enterprise Services

If Enterprise Services Architecture constitutes the principles that describe how successful IT systems can be built based on enterprise services, what of enterprise services themselves? To do our job of describing how to discover, design, and describe enterprise services, we must explain in greater detail the characteristics of these building blocks. This section will answer the following questions:

- What are the moving parts of an enterprise service?
- How do enterprise services fit into existing systems?
- Who uses or consumes enterprise services?

The condensed answer to these questions is this: An enterprise service is an **interface** to well-defined functionality provided by **functional business components**. The interface defines a set of **service operations** that perform work and can receive and send **messages**. Enterprise services are used by other programs that are said to **consume** the services. Services are **loosely coupled** to the entities that consume their functionality and to each other. The performance characteristics of an enterprise service, such as response time, are dictated by a **business service level**.

Let's now explain in a bit more detail the terms used in this condensed answer. Note that we will explain all highlighted terms, but the course of our explanation will sometimes require that we define them in a slightly different order.

Interface

A service interface describes what information can be sent to a service and what information the service sends back. Usually, an interface for a service has several distinct service operations that can be performed. Each of the service operations has a set of input parameters that are part of the messages sent to the service and then a set of output parameters that the service uses to send back the answer. One easy way of thinking of this is that the input parameters have the question being asked, while the service and the output parameters contain the answer sent back.

In our stopwatch service example, the start and reset service operations may be service operations that have no input parameters and return only a code indicating success or failure. The stop service operation may have no input parameters, but may return the time elapsed since the stopwatch was started. All of these service operations together represent the interface description.

In order to understand an enterprise service, it is important to understand what each service operation does. This means understanding the information sent in, any transformation that takes place, any persistent state that is changed, and the information sent back. The interface documents what is sent in and sent back. The service documentation explains any transformations and state changes. The service level agreement explains the standards for performance, volume of transactions, and other operational metrics that a service must meet.

It is worth mentioning as a technical implementation detail that enterprise service interfaces will be defined as Web services using open standards, in this case the WSDL. But this is of little concern for solution designers.

Messages

From a service design perspective, it can be useful to look at services from a message-oriented point of view. From this perspective, a service is simply an entity that receives and sends messages. The input parameters are one message, and the output

parameters are another message. This way of thinking avoids getting bogged down in implementation details, such as whether the service will respond synchronously or asynchronously to requests. The message-oriented point of view allows the service to be defined in terms of the essence of what it does, not how it is implemented.

The message orientation of services is a key factor in increasing loose coupling and making enterprise services and the systems built on them flexible and reusable. The message orientation of services refers to the fact that services are able to act as traditional functions that are invoked or can instead send and receive messages.

When a service is invoked like a function call, we assume that the response is passed back immediately, usually in a number of seconds. The service must have all information already needed or must have the information passed as parameters of the call to assure this immediate response. We call this behavior synchronous.

When a message is sent to a service, we cannot make any assumption of how long the response needs (could be in terms of weeks or months, depending on whether human interaction, workflows, or whatever may be involved) – the provider may be dependent on other results of other providers to deliver the response. We call this behavior asynchronous. Furthermore, there may be multiple responses logically related to one single request. Message orientation, then, provides far more independence in the relations between services than function calls do.

Later on, when we are talking about how to describe what services do, we focus on various patterns that describe different ways in which messages interact with services. There are three primary interaction styles – that is, patterns of sending and receiving messages – of interest to us as we are designing services, as shown in Figure 2.

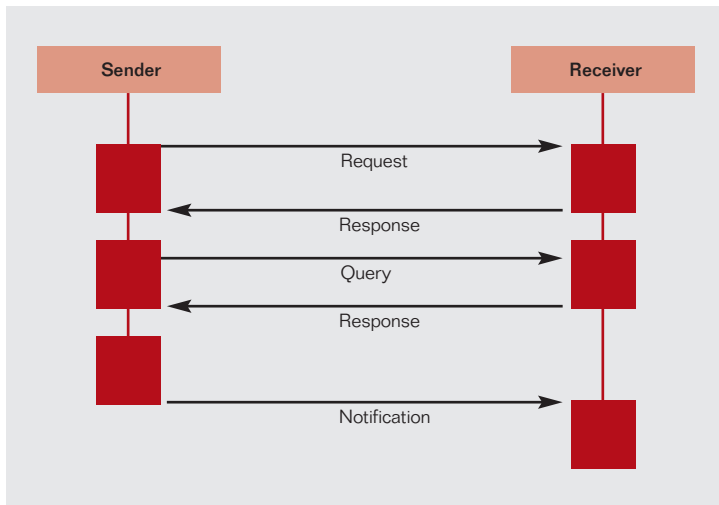


Figure 2: Interaction Styles

The patterns have the following definitions:

- Request/response means that messages are sent back and forth and that the state maintained by the service may change.
- Query/response means that messages are sent back and forth, but that the state maintained by the service will not change.
- Notification means that a service will send a message that contains notification of an event.

The following example illustrates the usefulness of the message-oriented point of view. Let's assume that we have defined a service that uses the request/response paradigm to open a purchase order and then respond when the credit of the purchaser has been approved. When we talk in terms of messages, we can explain the service in terms of the following messages:

- *CreatePurchaseOrder*: The initial message that contains the purchase order information.
- *CreatePurchaseOrderConfirm*: The response message that indicates that the purchase order was created and returns the number of the purchase order or any error messages about incorrect information.
- *BuyerCreditConfirm*: The notification message indicating that the buyer's credit is approved or denied.

Note that in describing these messages we don't have to specify whether the responses will be returned synchronously or asynchronously or how they will be targeted to the responders.

Loose Coupling

One of the most important features of enterprise services is the way in which they are loosely coupled to each other. In general, loosely coupled systems are made up of parts that interact in well-defined ways and know little of each other's inner workings. With respect to enterprise services, loose coupling means many different things:

- Loose coupling means that the internal workings of each service are hidden from all other services. The workings of a service may change in any way, as long as the behavior of the service that is described in the interface and the documentation remains the same.
- On a technical level, loose coupling means no dedicated request/response channel is established and kept open during the processing. Such connections are expensive in terms of system resources, may fail easily, and are difficult to reestablish (especially in between a request and response).
- On a logical level, loose coupling means the parameters and the created responses are transferred in the form of documents (based on XML) rather than a binary list of values. Since the documents are semantically marked up, they are much more tolerant in terms of the order and data types of the different fields or missing or extra values. The service provider component is unknown for the consumer, which means it is only known which operation is needed, but not who will provide the functionality behind the operation.
- On a business level, loose coupling means that processes are represented as much as possible as event-driven chains. A service consumer just publishes a certain business event even without knowing who is going to react to it – maybe even more than one might receive the event.

The opposite of loosely coupled is tightly coupled, which means that the internal workings of a system are exposed and that, to work with the service, you must understand the details of its internal workings. With tightly coupled systems, the level of complexity to make services work together or to make changes is very high. One of the goals of loosely coupled systems is to reduce this sort of complexity and make reuse much easier. Tightly coupled systems are difficult to change because the complex interrelationship of all the parts means that the effect of changing any one part is hard to predict. People quickly become afraid of changing tightly coupled systems.

The decomposition into loosely coupled components makes interaction between parts explicit and therefore transparent. There is a well-defined contract about the interchange of information between different functional areas, and change becomes something that can be accomplished easily and safely.

Service Operations

Each interface to a service describes one or more service operations that can be performed by the service. For extremely simple services, there may only be one service operation. Most services, like the stopwatch service, have several operations. Enterprise services are always a gateway to functionality provided by an existing system called a functional business component, which may exist or may be built from scratch. The relationship between service operations, services, and functional business components is shown in Figure 3.

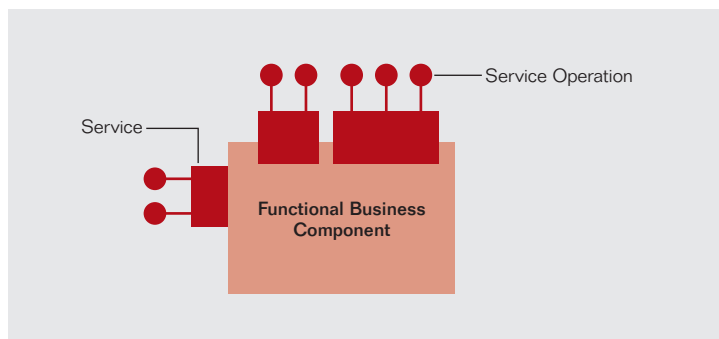


Figure 3: Services and Service Operations

Notice how each service may have many service operations and each functional business component may have many services. Functional business components are groupings of related business functionality like payroll accounting or general ledger. Functional business components usually correspond to the scenario level that was part of the solution maps described earlier.

Service Consumers

When a user interface or an application program or another service uses a service to perform a task, it is referred to as consuming a service. This language suggests that the functionality of a service is being provided to the user interface, application program, or other service. This language is slightly problematic from a message-oriented point of view, because when messages are sent to a service, it is possible to think of the service as consuming the input messages and sending back output messages. But this is not the way the terminology works. Invoking or using a service means you are consuming that service.

Business Service Levels

The quality criteria for a service in general is usually called the business service level. The operational characteristics of a service are described in the business service level, which describes the standards for response and operational load that the service will meet. When designing services, it is important to have some idea of the operational load that the service may have to satisfy because this may influence the design. For example, a service that will be under heavy load may be better implemented as a set of many simple service operations that can be load balanced, rather than fewer larger service operations that may be more difficult to partition.

To stress that, in the case of enterprise services, these quality criteria also exist (but cover a business functional level rather than technical attributes), we would like to introduce the term **business guarantees**. Business guarantees are descriptions of the behavior of the service not on an operational level, but on a level that is more meaningful for users of the service. These guarantees should describe exactly what can and cannot be expected from the service.

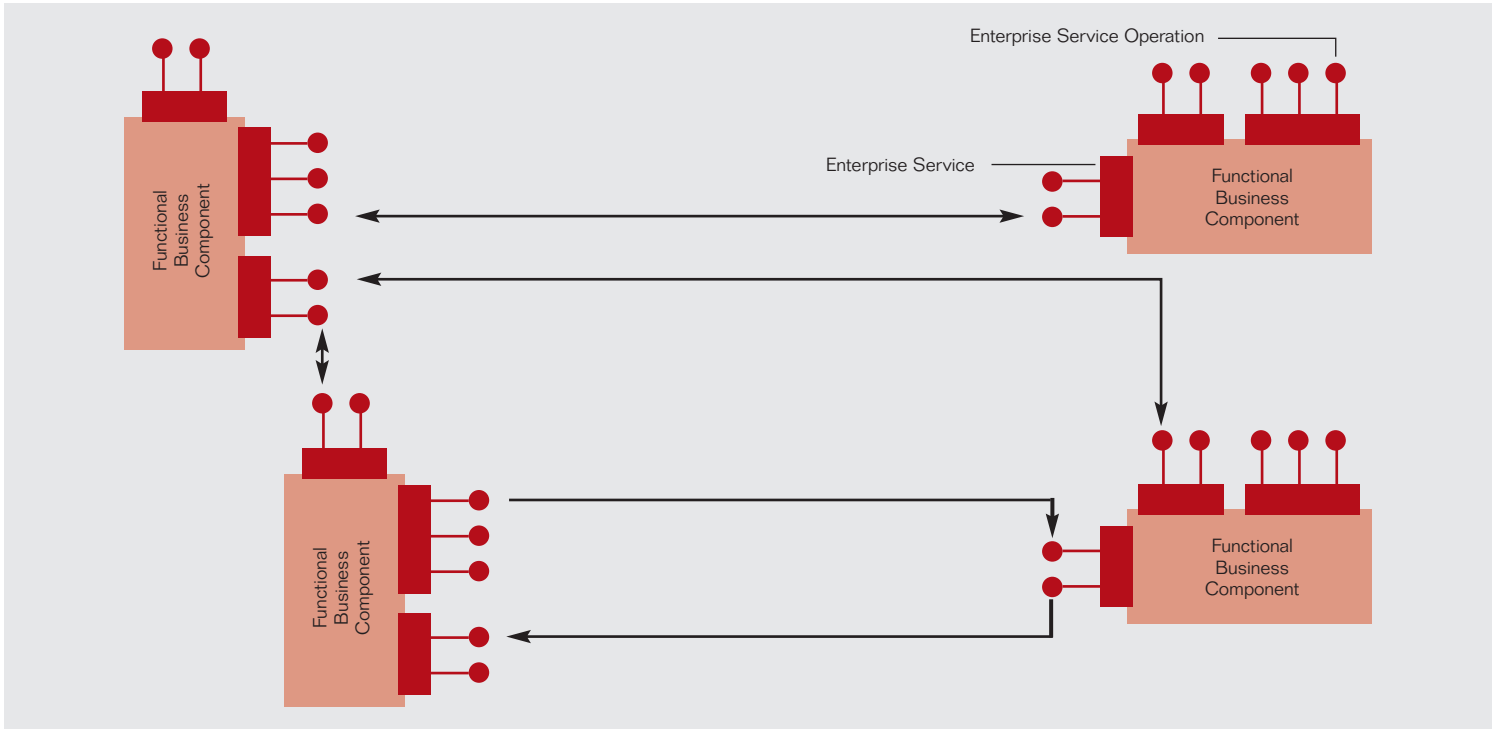


Figure 4: Functional Business Components, Enterprise Services, and Operations

For example, the specification of the enterprise service operation *placeOrder* simply describes how, when placing an order, it will be verified (does the receiver understand the order content?), evaluated, and, if possible, accepted (does the receiver accept the order and is the receiver able to fulfill it?). Once accepted, the *placeOrder* service operation will complete the order, which means the sender will receive the goods and (later) an invoice. It may also contain the assurance that the receiver will receive status updates on delivery dates and the actual shipment. On this level, there might be also guarantees about the maximum response time for order acknowledgment or order acceptance.

These times might be refined or parameterized for configuration according to the needs of a particular installation.

Functional Business Components

What makes an enterprise service different from a generic service is the scope at which it is defined. Enterprise services are designed to serve the needs of business processes at a level that provides for maximum adaptability and reusability, with the smallest amount of complexity in composition logic. This generally means that enterprise services are of a much larger scope and contain much more functionality than the typical atomic service.

The concept of a functional business component is important because it helps solution designers group related services together. SAP has a repository of functional business components that is used to keep track of which business processes are implemented in which solutions. The repository maps the functional business components, both to the business processes they automate and to the solution in mySAP Business Suite that implements them. Figure 4 shows how a set of services defined on functional business components might be related to each other.

Although enterprise services can be implemented in any way whatsoever, it is useful for a variety of reasons to model a solution using functional business components as a container for business processes and services. Besides allowing you to group related processes together, it also allows you to separate the model for how business processes are related from the way they are implemented and deployed.

For example, consider two functional business components, Payroll Accounting and General Ledger. Although both components are implemented by one piece of software (mySAP ERP), it makes sense to define a service interface between both of them, since it creates multiple deployment options. A customer might run one mySAP ERP representing the General Ledger, whereas in each region other instances of mySAP ERP are hosting the Payroll Accounting functional business component instances. Note that, when partitioning work across a group of services, it is important to have an unambiguous method for assigning work among the multiple services.

Here's another example. *CreditManagement* can be understood as a functional business component, *CreditLimitMonitoring* would be an enterprise service, and *Approve-CreditLimitAvailable* or *ExpandCreditLimit* would represent operations of this enterprise service.

Business Drivers

For the purposes of enterprise services design, it is useful to boil down the vast topic of Enterprise Services Architecture to its essential benefits. By doing so, we are able to quickly evaluate the potential value of ideas for enterprise services in the same way, which makes it easier to compare and prioritize our work.

Enterprise services generally provide business value in one of four ways:

1. Improving user productivity and ease of use
2. Enabling support for innovative new business practices
3. Increasing automation and efficiency of business processes
4. Allowing more flexibility in deploying an application

While each of these business drivers provides its value in a different way, it is important to note that enterprise services that provide any of the first three benefits all have the following effects on a business. Enterprise services that improve user productivity, support new business processes, or increase automation tend to:

- Speed cycle times for execution of business processes
- Reduce human errors
- Reduce costs

Enterprise services that provide these benefits also provide unique value, which we will now discuss.

User Productivity and Ease of Use

Some ideas for enterprise services will be particularly good for helping create solutions to make users more productive by making it easier to build role- or task-specific user interfaces. For example, let's imagine that a task that requires a user to enter data on 10 different screens can be reduced to 1 screen because of a service that collects the data and then enters it using the proper transactions into an enterprise application. Users could then do their work in a fraction of the time, and they would be much happier as well. The easier a user interface is, the more people can use it and the less training it requires. When a user interface is transformed from something that is highly technical to something that is simple and easy, more people within a company tend to use it, which expands the amount of value a company gets from an application.

Or perhaps the simpler user interface can now be exposed to outside partners, suppliers, or customers who can enter data themselves or check on the status of various processes. Maybe each type of user has a special interface to meet his or her needs. This sort of transparency and self-service may reduce time that is wasted handling unnecessary phone calls or e-mails requesting information. Ease of use plays a big factor in keeping users satisfied and making them loyal to an enterprise application instead of hating it.

Remember that the enterprise service doesn't create a new user interface. Instead, enterprise services focused on user productivity allow new and better interfaces to be created. One service or set of services may enable many different roles to have their own user interface. This is enabled by the separation of user interface components from business logic. This decoupling allows companies to reuse the business logic to a large extent, which makes it easy and much cheaper to implement many role- and task-specific user interfaces. Without this decoupling, there is always the attempt to do as much consolidation as possible and reduce the number of user interfaces to reduce the effort of implementation and maintenance, which has a dampening effect on productivity.

Next Business Practices and Process Innovation

Another key way enterprise services help is by supporting new and innovative business processes that would be impossible to support without the services. Innovations like a rapid, available-to-promise capability are sometimes called "next practices" because, if successful, they will be imitated and become best practices. In other words, they represent the next wave of best practices and are therefore called next practices for short. Before such a service existed, when a new order arrived, someone would have to check inventory and production schedules and call around to make sure capacity and sufficient parts to fill the order was available. This is a process that might take a few days. With an available-to-promise enterprise service that checks all the important details at the warehouse, factory, and suppliers, the sales staff may be able to answer a customer inquiry and commit to fulfilling an order while the customer is still on the phone, which could reduce delays and increase sales. Services that support next practices frequently have the effect of bringing IT into precise alignment with the needs of a business. Existing processes supported by an application can stay in place, but services built on top of applications allow enhancements.

Business Automation and Process Efficiency

Enterprise services frequently enable IT to completely automate a task that has been previously only partially automated or to attack particularly annoying areas of inefficiency. Sometimes these services provide missing functionality or solve an integration problem that has not yet been addressed.

A simple example of a service that increases business process automation and efficiency could be a service that helps streamline order processing. An enterprise service could help automate data entry by providing an interface to a scanner, checking the data, and then entering the order in a system. If possible, the service could attempt to check whether all information needed to respond to the order is available, and, if so, automatically complete the order. Such services, while not always complicated or glamorous, frequently make a huge difference in cutting costs, increasing efficiency, and providing better quality, more consistent service to customers.

Deployment Flexibility

Enterprise services that help deployment flexibility of an application are a bit different from those that provide the other three benefits. Enterprise services that enable flexible deployment increase the choices available to those responsible for managing an application. This increased choice can have many different sorts of benefits associated with it.

For example, if the process of submitting expense reports is supported by a set of services, it may be possible to use services to redirect the flow of expense reports in a variety of ways without having to change the way reports are submitted. Perhaps the expense reports could be directed to an expense clerk in each department who might process the reports manually. Or perhaps they could be directed to a central staff in the company who would use an automated system to process the reports. Or perhaps the reports could be directed to one or more outside processing firms located offshore. Because services hide the actual processing of the expense report from the user submitting them, all of these options or a combination could be pursued.

Functional Categories of Enterprise Services

Categorizing enterprise services by their business value is one way of evaluating their potential impact. Another important way to look at services is by the functional role they will play in a system. This perspective provides a window on how and where a function will be reused. Looking at both of these dimensions together, which we do at the end of the section, provides even more insight into the nature of an enterprise service.

There are four categories for the functionality of an enterprise service:

- **Component services** keep track of the context – that is the relationships, data, external information – related to an important business function.
- **Process services** trigger a process and help manage the consistent execution of a process.
- **Entity or engine services** model and provide access to a business object or engine. An entity service might represent a create order, which makes sure that every time an order is created, all business rules are checked and all data is consistent. An engine service provides access to commonly used functionality like a pricing or tax calculation engine.
- **Utility services** perform some commonly used functionality for other services or for consumers of services. A service that provides the allowed values for a specific field (a so-called value help service) is a classic example of a utility service.

When discovering and designing enterprise services, keeping the functional categories in mind helps the process in three ways. First, asking which functional category a service falls into can help us clarify the purpose of the service. How will this service interact with the other related services? In practice, most services play a role that falls clearly into one of the functional categories. When a service starts to play two functional roles, its scope may be too large and perhaps it should be split into two services.

Functional categories also help with brainstorming to discover services. When we are looking at a business scenario and attempting to conjure up ideas for services, we can start by asking the following questions: What component services might be needed? What process, entity, and utility services would be useful in automating this scenario and helping automate related processes?

Finally, when an idea for an enterprise service has occurred to us, we can flesh out and complete the idea by using the functional categories as protopatterns. If an idea for a service appears to be an entity service, it should perform certain functions. We can then ask ourselves what those functions should be as a way of completing the service. Most importantly, on what level do we want to enable flexibility? Do we want to use services for creating and accessing entities, or do we want to move process steps around? This is the fundamental question, to which there is no simple answer. Different needs in different situations for different organizations will determine the appropriate priorities.

To use functional categories in any of these ways, they first must be more precisely defined.

Component Services

A component service is probably the most difficult to understand because its function is to keep track of relationships that are not simply represented by a data field. For example, a service that provides access to the credit limit for a customer and his or her existing balance is a classic component service. The service would provide not only the credit limit and balance, but also access to external providers of credit ratings, information on currently unpaid invoices, and payment history. Such information could be provided by other services consumed by the credit management service.

Process Services

A process service triggers the execution of a process or a process step and then manages the execution of that process. Process services assume control of execution when called and then orchestrate services or other functionality to carry out consistent execution of a process.

For example, **closed-loop order processing** allows a company to receive orders from its customers electronically and use the data and functionality of its enterprise resource planning (ERP) application to automatically respond to the order whenever possible. Automating the process provides the biggest benefit, but in exceptional cases in which no response can be generated, a manual workflow needs to be started in order for a human to handle this order. To completely solve this problem, a closed-loop order processing service must have a service level that guarantees that either the order will be processed automatically or a manual workflow will be started. This is very different from simply offering a service that guarantees that an order is successfully received. The interface may be identical, but the underlying application functionality is more complex and challenging.

Entity or Engine Services

Entity or engine services provide access to the data and functionality of an entity (typically a business object) or to an engine (a complex calculation of some sort). An entity service might guarantee that a business object like a purchase order is properly created according to all the business rules for data completeness and validation. An engine service might calculate shipping rates for packages being sent domestically and internationally. Two classic examples that can help us illustrate some of the design challenges are a sales order and a pricing engine.

- **Sales order as an entity service**

A service enabled sales order entity may offer some basic services to create, retrieve, update, and delete an order. It should also offer services for specific functions that are commonly used in business processes (for example, changing the delivery date).

- **Pricing engine service**

A service enabled pricing engine may offer a service to calculate prices based on some set of input data, as well as all services necessary to maintain pricing conditions.

Both entities and engines can in turn have dependent entities and engines. When we define a service, we need to think about whether or not to include these in the scope of our service. For example, here are two natural extensions of the sales order entity service and pricing engine.

- **Sales order entity service with delivery document**

A sales order can have an associated delivery document. If we want to offer a service that creates or updates an order, we have to decide whether it is necessary to also create or update the delivery document. Either way, this needs to be documented and made explicit to the consumer of the service.

- **Pricing engine with tax calculation**

The calculation of the price may or may not include tax calculation.

In many cases, it may not be necessary to completely service enable an entity or an engine. It may be enough, for example, to offer retrieval services, in which case we would expect the data or parameter input to happen through a traditional user interface. Here are some examples:

- **Sales order entity service offering retrieval only**

In a business with only a few but very complex orders, perhaps associated with large projects, it may not be important or meaningful to offer a service to create these orders. However, there may be specific functions, such as retrieving the delivery date of each item, that should be offered as services.

- **Pricing engine that performs calculation only**

A pricing engine could, for example, offer a service to calculate a price at run-time but not services necessary to submit prices and other conditions, in which case the latter would have to be maintained through a traditional user interface.

Utility Services

In any application, many services have very little business meaning on their own, but are convenient to have in order to build a user interface or to help integrate two applications. Examples of such services can be valuable help for a specific field, checking that the input format is allowed or as an authorization check.

Business Drivers and Functional Categories

Business drivers and functional categories are powerful tools for the solution designer who is attempting to create enterprise services. They provide a framework against which an idea for an enterprise service can be evaluated and quickly brought into focus. Just by asking which business driver is served by an idea or which functional category a service belongs in, a solution designer is able to focus on and sharpen ideas.

It turns out that early experience with enterprise service design has revealed a correlation between business drivers and the functional category of the services needed to realize the desired business. Figure 5 shows this relationship.

Business Benefits / Service Scope	User productivity and centrality	Next business practices and process innovation	Business automation and process efficiency	Flexible deployment and platform
Component				
Process				
Entity/Engine				
Utility				

Figure 5: Correlation Between Business Drivers and Scope of Service Enabling

The correlation is that each business driver is best implemented by certain types of services. This finding can be used to further speed the evaluation of an idea for an enterprise service, because it outlines one important dimension for services that are likely to be successful. Here is the way in which the services in each functional category meet the needs of each business driver:

- To increase user productivity, we will often build a new user interface, such as a work center or an interactive form, that is supported primarily by services provided by entities and engines together with the corresponding utilities. Process services are less important because the user is driving the process.
- New processes will often be supported by composite applications. Just as with new user interfaces, here we have a user who is in control of the process, so again services provided by entities and engines and the associated utilities are needed. A composite application may also trigger a process and hand the control over to the next application, in which case we need a process service.
- For process automation, we typically have one application calling another in more of a peer-to-peer situation, in which the control is handed over to the called application. This falls squarely in the domain of process services.
- Deployment flexibility is focused on managing the execution of business objects and processes represented by components, although an individual entity or engine service might be needed in some cases.

Please note that business benefits – and consequently the scope of the services defined – are not mutually exclusive. Analysis of one business benefit could lead to requirements that might produce any sort of service. These correlations are not meant to be hard-and-fast rules, but rather a way to accelerate understanding the value of an enterprise service and creating a precise definition.

The foundations of enterprise services provide a description of the arena in which a solution designer will play the game of solution design. The next section describes some of the significant challenges that will be found in that arena along with suggestions for how to overcome them.

Discovery and Design of Enterprise Services: A Partitioned Approach

It has taken three sections, but the preparation is completed, the context is set, and the main event can begin. Now, with everything necessary in mind, we can accomplish the core purpose of this document – to accelerate your progress in learning how to design enterprise services.

Our purpose is to educate, and our posture is humble. The methodology we will present is interesting and powerful, but it is certainly not the last word on enterprise services discovery and design. In fact, it is more like the first word. And we want to hear from you about how to improve this methodology. Please join the discussion on SAP Developer Network (<http://sdn.sap.com>) at the Enterprise Services Architecture section for the Enterprise Services Architecture preview pages.

The approach we take was carefully crafted to address all the people who will be involved in designing enterprise services. Some organizations will want to start small by designing one service, and others will be designing systems of services to work together. The most complex design task will be the challenge of service enabling an entire landscape of applications in a company or in a suite of vendor products. People in each of these groups will find the guidance that they need in the following sections.

The Discovery, Design, and Documentation of Enterprise Services

Regardless of the nature of the design task – whether one or many services are being designed – the process proceeds through the same three macrostages in preparation to hand off to those who will implement the services:

1. **Discovery:** First, a business-driven analysis reveals the scenarios that might benefit the most from enterprise services. Pain points, opportunities for creating value, and the business drivers of Enterprise Services Architecture are used to find promising scenarios. The business processes and the existing supporting solutions for these scenarios are then analyzed to determine what the requirements would be for services that would improve support for the business.

2. **Design:** Once a promising business process has been identified, the design process begins. The requirements may be satisfied by a single service, a system of services, or a program of service enablement. In any of these cases, a design must be invented, evaluated, and refined to meet the requirements outlined in the discovery process. This is usually an iterative process in which the design work improves the understanding of the requirements, which may then be changed in ways that affect the design. As we shall see in the next section, different things must be kept in mind, depending on the level at which the service design is taking place.
3. **Documentation:** The final step of the methodology, which involves properly documenting the new services and entering them in the enterprise services repository, will be described in section 5, which offers various templates for properly documenting the services.

With these steps accomplished, it is then possible to move to the implementation stage with the confidence that the services that are created will actually meet the requirements according to the dictates of the design.

The Three Levels of Enterprise Service Design

In order to serve the needs of everyone who is likely to participate in the creation of enterprise services, we have partitioned the design process into three levels, each depending on the one below it, as shown in Figure 6.

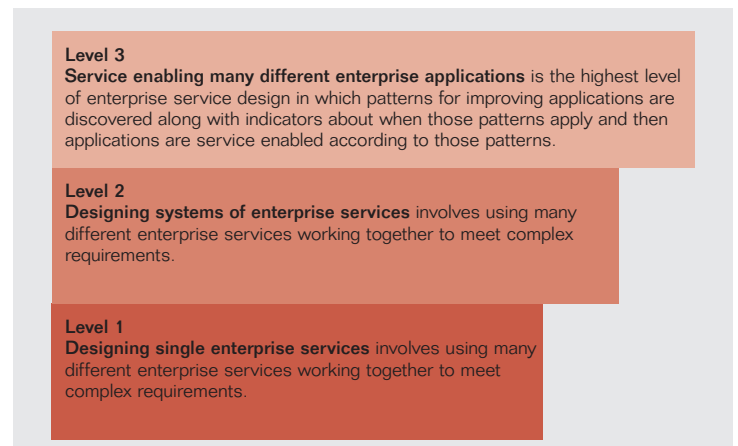


Figure 6: Levels of Enterprise Service Design

Designing services at level 2 or 3 is impossible without the one below it. Each level addresses a different sort of challenge that may emerge out of a discovery process. Sometimes, analysis of business processes will require design of only one service to meet the requirements. Other times, a system of services will be required. The most ambitious level is of course service enablement, where the design challenge is to improve the ability of many different applications to serve business needs by creating enterprise services.

Level 1: Designing a Single Enterprise Service

A single, well-designed enterprise service can be a powerful tool to meet many different goals. For example, information that is spread across many systems can be collected by one service so that solution designers can easily build a single screen to provide this information in a consolidated form for users. Such a service can frequently be a significant boost to productivity, because users of the system no longer have to gather information from many different locations. There are many other such opportunities that can be easily implemented and represent low-hanging fruit for solution designers.

Analyzing the most important scenarios in search of ways in which a single service can improve a business process, can be a great way to get started and build skills in discovering and designing enterprise services.

Level 2: Designing a System of Related Enterprise Services

After a clear understanding of enterprise services has been gained, more complex business processes may be supported by systems of services working together. For example, at some businesses the available-to-promise (ATP) service may be just a single process step, but for others it may be supported by a system of services all working together.

ATP might be a single process step for a manufacturer with one location and a small number of products. The internal systems could keep track of all information that would be required to answer an ATP inquiry, and one service could consolidate information to calculate an answer. At most companies, however,

ATP would be implemented using some services that query internal systems across many plants and other locations as well as other services provided by partners giving insight into raw materials, inventory of work in progress, production schedules, and other needed information. Partners might then use services from their raw materials suppliers or logistics partners. Such cascading may be two or three levels deep.

The requirements met by systems of services vary widely, but usually involve creating a set of services across the functional categories, each designed to address a different aspect of the needs of a business process. Component services solve part of the problem, with process, entity, and engine services all playing their role. It is the solution designer's role to craft these services not only to serve the needs of the process context, but also to be maximally useful as individual services. It is not uncommon for the design of a system of services to result in a redesign of the business process being supported.

The key to creating a useful set of services is to understand how all of them will work together to support a business process and how each may be used separately to support many different business processes. If a certain activity reflects one process step from the business point of view, it can and should be reflected as one service (especially if it is more or less identically reused in different processes) although multiple technical steps are required to perform it. On the other hand, if the activity can be decomposed into different substeps that still have an individual business meaning and may be useful to support other services or business processes, multiple services are a better design choice. Making these decisions so that later designers of composite applications find exactly what they need to solve their problems easily is the true art of enterprise services design.

Level 3: Service Enabling Many Different Enterprise Applications

Once an organization understands how services work individually and how to create systems of services, a program of service enablement can be initiated. The goal of such a design program is to undergo a large-scale analysis of many different systems and systematically add services to them.

Controlling such a design program so that the resulting services are well designed and serve to improve each system and the ability of all systems to work together is an enormous challenge. SAP has developed a method of coordinating the activity of solution designers and distilling the best thinking by using a system of design contexts and indicators that suggest when a design context might apply. Service enabling can take place at any size organization, although the full-blown approach described later is most frequently used at the largest organizations or at commercial software companies that will be able to make a huge leap forward through service enablement, which justifies the investment in such a program.

Design Contexts and Indicators for Enterprise Services

In one sense, each design of an enterprise service is a process that must invent a solution to meet a unique set of requirements. But in its experience in creating hundreds of services for mySAP Business Suite and the SAP NetWeaver platform, SAP has discovered various patterns of transformation that are part of the service enablement process. SAP calls these patterns **design contexts** and has developed a set of indicators for each that helps determine when that pattern might apply.

Design contexts were invented to help with a program of service enablement, but it turns out that they apply to all levels of enterprise service design. Remember, methodologies exist to help people do things over and over again with consistency and efficiency to achieve a degree of high quality. When we think of something like designing enterprise services, which could be created to serve almost any purpose, we must focus our thinking, if we are to be able to craft a methodology that is at all useful and contains the results of the experience gained so far in designing enterprise services.

So far, several ideas have been introduced to focus our thinking:

- The business-driven approach of solution maps shows us how to understand a business context, sort out the related groups of scenarios, and focus in on a business process.

- The Enterprise Services Architecture business drivers tell us which sorts of business value enterprise services are likely to create.
- The functional categories reveal common patterns for services.

The next way we will focus our thinking is to use the concept of the design context, which should be thought of as a pattern for improvement that falls under one of the business drivers. So far, SAP has discovered design contexts for user productivity, process innovation, process efficiency, and improving deployment flexibility.

A design context is a pattern for a common way of improving an application by adding enterprise services. In the user productivity areas, there are several ways that have been found to help users interact successfully with computer software. One of those design contexts involves using a work list–oriented user interface design pattern to provide users with self-service functionality and reduce dependence on experts. The design context for this improvement shows how a series of user interface elements can be used in various types of screens to enable this work. In other words, the design context is a sort of template for a successful user interface that can be supported by enterprise services. A design context explains a lot about how the enterprise services should be designed, which should help achieve the goals of our methodology to increase productivity and quality. A design context is not a design, but rather the shape of a design that may apply over and over.

Indicators are features of a business process that suggest it might benefit from a solution based on the design context. Each design context has its own indicators. One of the indicators for the work list–oriented design context is when an expert must be frequently consulted by a larger group of workers to do certain tasks. This indicator suggests that some sort of self-service could help. We will explain this and many other indicators later. The point is that indicators help determine whether existing business processes are appropriate candidates for a design context.

It is easy to see how valuable design contexts and indicators could be part of a process of service enablement. With a set of design contexts defined, solution designers who are analyzing business processes and looking to discover enterprise services can use the indicators to quickly determine if a design context applies. If it does, by applying the design context, the work of creating a set of enterprise services is accelerated.

Now it is clear why our methodology is the first word on this subject. It is quite clear to us that many more design contexts will be discovered in addition to those we have included in this document. Once discovered, many new indicators will lead the way to them.

Even if you are designing just one service or a system of services, we are confident that your work will be improved by understanding the design contexts and indicators, which will provide clues and examples for successful enterprise services.

There is more to say on this topic, and it is covered under the section “Creating Your Own Design Context and Indicators” at the end of this section.

Level 1: Discovery and Design of Single Enterprise Services

During the discovery process, as scenarios and business processes are analyzed, requirements for enterprise services will emerge. The question that designers of enterprise services must answer is, “How can services help?” The first step in our methodology is to introduce a set of indicators that suggests how services might be beneficial.

The next question that faces designers is, “Can I meet these requirements with a single service, or will many different services be required?” This is not a question that can be answered easily at first, but as a solution designer gains experience, the answers come faster. A surefire way to answer this question is to draft a design for meeting the requirements with a single service and

then examine that service to see whether it meets the design criteria for high-quality, repurposeable enterprise services. This section also offers guidance about what a well-designed service looks like, which should help in evaluating ideas for services and determining if one or many are needed.

Indicators for Enterprise Services

By keeping these indicators in mind during the discovery process, a solution designer will be able to quickly recognize opportunities for applying services. Remember, these are not the only indicators of services.

***Indicator 1:** Complete processes or process steps with frequent high-execution time or frequent high-execution costs are potential candidates for being supported by services.*

This indicator recommends following the money and following the time. If something is taking a long time or costs a lot of money, perhaps it can be done better by adding services to streamline the process.

***Indicator 2:** Information logically linked and often needed by users is distributed throughout disparate systems.*

This indicator recommends applying services to prevent users from having to look in many different screens, in many different systems, for the information they need. A service can do that just fine.

***Indicator 3:** Business know-how is decentralized, but decentralized users need support from central users to get their business done. There is a high call or request rate to expert users that results in repeated information retrieval from application systems or the application of well-defined business rules (such as, tasks that could be automated).*

This indicator recommends creating a service that models, to the greatest extent possible, the knowledge of an expert. When users need to consult the expert, they can use the service first, which may provide the needed assistance.

Indicator 4: Information is manually transferred from paper into applications or between applications.

This indicator recommends using services to transfer information to avoid manual processes.

Indicator 5: Your research shows a white space, a need for automation that is not filled by your current portfolio of enterprise applications.

If the white space is an end-to-end scenario, a potential new composite or business renovation opportunity exists.

This indicator recommends that, when an opportunity for extending automation appears, the need should be filled through a service or group of services designed to work together as a composite application, an application created by assembling services to work together.

Indicator 6: You find that a “standard” application is highly customized or modified. This is an opportunity to rearchitect and renovate your standard application into a more flexible, adaptive composite application.

If an application has been extended or enhanced through configuration or custom code, this indicator suggests that services may be able to support the same need in a more flexible manner that reduces the cost of evolving the application.

Indicator 7: All needed input parameters for performing a specific activity are already available in a system, not necessarily in the system implementing this activity.

This indicator suggests that automation can be extended by using a service to gather inputs that may be available already in some form to carry out a specific activity instead of assembling the inputs manually.

Indicator 8: Important data from external (technical) sources or functionality not supported by your existing enterprise applications is either needed within the process in general or enables the automation of the respective task.

This indicator suggests that services can assemble information from external sources or provide functionality that is needed to extend automation.

Indicator 9: In some cases, an application might reach a “steady state” although a process is not completed (such as, the application has finished processing data, but has not triggered the events that logically follow in the business process). In such cases, manual intervention is usually necessary to get the process going again.

This indicator suggests that a service can monitor a process and detect when the conditions are satisfied for the next step of the process and then trigger continuing execution.

Indicator 10: A process is crossing a (potential) borderline to other applications, other systems, or external organizations.

This indicator suggests that services can manage the processes that cross boundaries between applications, systems, or companies.

Indicator 11: Industry standards (for example, CIDX, RosettaNet) can be an indicator for services that optimize the communication between applications and industry standards.

This indicator suggests that services can play a translating role between industry standards and applications.

Indicator 12: A given business process needs external synchronization to a business-to-business (B2B) scenario without including details of a specific standard.

This indicator suggests that services can manage B2B processes, allowing applications to participate in such processes without modification to add support for the process or for an existing standard.

Indicator 13: There are different variants of a given business process in which major parts are stable and only single activities may vary.

This indicator suggests that if a process is implemented in many different applications with only a slight variation, then services may be able to implement the process once and still accommodate the variation.

Indicator 14: A solution or application often receives data from various other solutions.

This indicator suggests that services can route information between applications.

Indicator 15: *A solution often is deployed multiple times within one company, with data exchange between the solutions.*

This indicator suggests that a service can encapsulate a replicated solution so that where and how the solution is implemented is no longer visible to end users. In this way, the solution can be implemented centrally or distributed as needed with minimal impact.

Indicator 16: *Check the relevance of business process outsourcing (for example, whether business process outsourcing (BPO) providers offer outsourcing services for a given business process, or whether customers are asking us for interfaces to BPO providers). The outsourcing borders, interfaces, and messages exchanged between the BPO provider, enterprise, and third parties are indicating the need for services.*

Services can be used to encapsulate processes that can be executed externally to an organization by BPO providers.

It is important to use these indicators as intended – as clues to the right design context or right structure for a service – not as rigid rules. Indicators at this point cannot be used this way because they are general and overlap.

For example, indicators 2, 4, 7, 8, 11, and 14 all seem like instances of a common pattern of using services as a organizing and synchronization mechanism for information that is located in many different places. Each is slightly different, but they are talking about different aspects of a similar design challenge.

Within this group, indicators 8, 11, and 14 are all targeting the transmission of data between systems, but with the following distinctions:

- Indicator 8 relates to more technical-driven services that might be used for devices and embedded systems.
- Indicator 11 reflects common B2B protocols that need entry points on both sides of the business partner to post the needed messages and events.
- Indicator 14 describes more the classical application-to-application integration issues among enterprise solutions and with other external applications.

These indicators point to so many fascinating design challenges that this analysis could continue for pages. The point of indicators is not to analyze them for their own sake, but to use them as a way to accelerate enterprise service design.

At the end of the discovery process, with the help of these indicators and an understanding of the business processes, a solution designer should then have a list of issues that could be addressed by services. Now the challenge shifts from requirements to design.

Single Service Design Guidelines

Let's say that a business scenario has been analyzed and that two or three indicators are present. How many services are needed? As we have said, this is a question that will be answered more easily as a solution designer gains experience. But at first, one way to answer the question is to imagine that just one service will be created to satisfy the need. We can then evaluate that idea for a service against the criteria for a well-formed service below. If the initial design meets most of these criteria, then perhaps a single service is the right choice. If it does not, maybe a system of services might be needed.

Here are the guidelines for designing a single service and some questions to ask to determine if an idea for a service has characteristics that will lead to success.

Be sure to understand how a service will be used. One of the easiest ways to stay on track when designing services is to adhere to the “provide and consume” principle. When providing a service constantly, imagine how it will be consumed. Who will use the service and for what purpose? Think of as many examples as possible of how the service will be used and write them down.

How will the service support business processes? Related to the provide and consume principle is an understanding of how a service will play a role in supporting one or more business processes. The most reusable services are those that provide a chunk of functionality that can assist in implementing as many different processes as possible.

Can a service ever be too small? Some services are really useful and are quite small in scope. One can imagine a service to get the date and time as such an example. It is hard to say in general when an individual service is too small, but if the orchestration logic for services starts to become quite complex, it may be an indication that one or more services is too small and should be redesigned with a larger scope.

When is a service too big? A service may be too big when it has a huge parameter list and has many different purposes. To reduce complexity and provide reusability, services must have a clear identity. When a service becomes a catchall or serves too many different tasks, it may need to be divided into multiple services.

Can a service be made more useful? While a service should not be too big, you should attempt to address as many needs as possible. If extending a service helps it play a role in many more business processes, the extension is probably a good idea.

What business driver will be satisfied by this service? One way to help understand the identity and focus for a proposed service is to ask: which of the business drivers of Enterprise Services Architecture is being enabled by the service? While it is possible that a well-designed service can enable more than one business driver, most frequently a service promotes just one of them.

What functional category service does this seem to be? Well-designed services most frequently fall into one of the functional categories for services. If a service is a hybrid without a good reason, it may need to be divided into two services.

Can the number of service operations be reduced? In general, the best services do their jobs with a small number of service operations. Frequently, this number is 4 to 6, but small is a relative term and 4 to 6 could be too large as well. A solution designer should always be asking whether too many functions are being provided.

Can loose coupling be increased? Loose coupling is a key benefit of services. The less each service needs to know about the other services around it, the better. The less each service needs to know about how a message is being sent to and from it, the better.

Increasing loose coupling means that a service becomes more general purpose. In general, the less dependencies that each service operation has to previous service operations, the better. Isolation of services (components) means that they have to know only what messages are exchanged, but not how they are processed. In an ideal world, even the receiver of the message is not really known by the producer. In terms of services, it means the service consumer does not really know who the provider is. As loose coupling is increased, applications become more and more stateless, like HTTP.

Will the service ever be part of a system of services? If a service is going to be part of a system of services, clues to its proper identity can be gained by asking what role it will play and what role the other services will play.

By asking these questions, the design of most proposed services can be improved. The next section outlines different scenarios for systems of services.

Level 2: Designing a System of Related Services

As soon as it is clear that more than one service is needed to satisfy the requirements, you have a system of services on your hands. Systems of services as a group should have most of the same characteristics as individual services: a clear identity, loose coupling, the right scope, and so on. The individual services in a system of services must adhere to the guidelines for single services.

The question for this section is how can groups of services work together? This is a challenging question that really has no general answer. We can provide some assistance, however, by explaining some common requirements that systems of services frequently have to meet and how they can be met. In this section, we cover three issues that arise over and over in creating systems of services:

1. Support for transactionality in an informal sense (such as, functions equivalent to the commit and rollback of various activity)
2. Support for analytical processes
3. Management of master data

Experience in early design of enterprise services has revealed some ways to meet these challenges, which are described below.

There is another major role for systems of services that is general but quite important, and that is the role of maintaining relationships between business objects under the ownership of different functional business components. The relationships between the business object owned and used by ERP are contained within the ERP application. In a world of composite applications that use services and business objects from many different sources, relationships between objects that may be owned by ERP, customer relationship management (CRM), HR, or supply chain management (SCM), but that are used together to solve a new business problem, must all be tracked by a system of services. Maintaining such new relationships is one of the key ways that composite applications extend what is possible with a set of existing systems.

Business Transactions

Most business processes are designed to handle mistakes, errors, and cancellations elegantly. From the database level up, the general idea of being able to start a process, but then roll back all work if something goes wrong, is referred to as support for transactions or transactionality. At the level of a database, basic transactional support involves opening a transaction, meaning taking a snapshot or creating a checkpoint that preserves the current state, then doing work, then either committing the transaction, making the changes permanent, or aborting the transaction and rolling back the changes to the checkpoint. But business transactions in a distributed environment are more complex than for a single database. A business transaction typically spans a long period of time – days, weeks, months, or even years – far longer than the typical database transaction, which is opened and then quickly closed in a matter of minutes. In such situations, a transaction monitor is employed that keeps track of all of the transactions in all related systems and then commits or rolls back the changes in each as needed. The challenge facing solution designers in designing systems of enterprise services is more general still. Systems of services need to provide ways of undoing operations that may be aborted during execution of a business process, a task similar to formal transactional support, but fuzzier.

At the outset, it seems that this could become a nightmare of complexity, but there is a simple way to handle informal transactions at the level of the service operation. Whenever a process needs a way to achieve transactional support of some kind, create service operations that express this functionality at a high level. One service operation can express the beginning of the transaction, and two others can express committing or rolling back, if these operations need to be exposed. Sometimes only rolling back needs to be exposed because opening and committing the transaction happens as part of the normal process. These service operations needn't be implemented with commit and rollback on a database level, but rather they can do and undo the work to restore changed data to its original state, but at the same time keep track of what almost happened to meet auditing requirements.

Figure 7 shows a set of services in a business process with a *revokeDetailedAssortmentPlan* service to do the rollback and a *releaseDetailedAssortmentPlan* to do the commit.

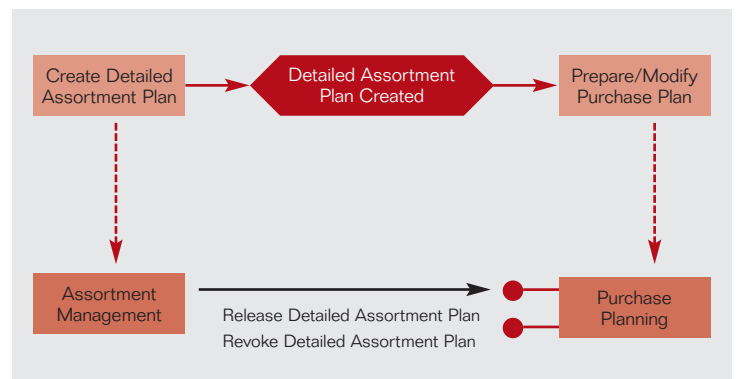


Figure 7: Compensation Operations

Analytics

In the world of enterprise services, data and functionality are necessarily hived off into separate services that can easily be combined and recombined to solve new problems. One of the challenges of creating a system of services for such an architecture is how to bring the data back together, so it can be used for analysis and reporting. Too frequently, solution designers ignore this need, assuming that somehow the data will be consolidated

through some extract, transform, and load (ETL) process against the underlying database. This is a significant mistake, because it creates a situation in which the process of assembling data for analytics becomes costly and implementation dependent.

The simple solution to this challenge is to create sets of related services that anticipate the need to assemble data for analytics. An analytics and monitoring service of the sort shown in Figure 8 has service operations that allow all the other services to send data. This structure maintains the information hiding and loose coupling, which are huge drivers of value for Enterprise Services Architecture.

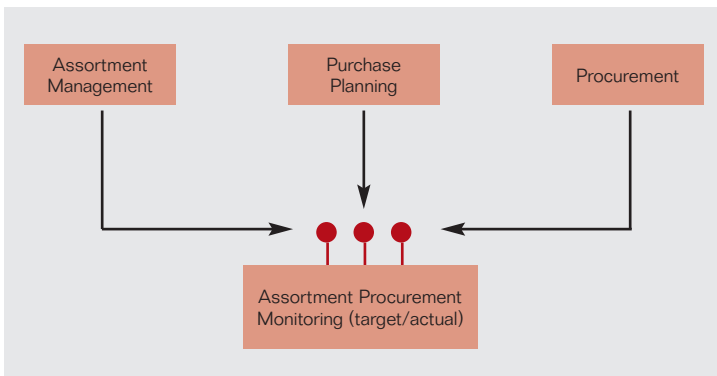


Figure 8: Using Enterprise Services to Support Analytics

Master Data

Master data management is the opposite problem of analytics in terms of the distribution of information. Frequently, in modern enterprise applications, master data for customers, products, and other frequently used information is maintained in centralized hubs to ensure quality control. Applications that use that data must subscribe to it so that the data can be distributed to them or so that they can access the data in the central hub through an entity service. Distributing the data may make sense if a service needs rapid read-only access to the data and does not want to invoke a call to a central service or if a large amount of reference data is needed by a service.

If data will be distributed, solution designers should make sure systems of services include service operations that allow master data to be distributed to the services that require the data. Figure 9 shows how a product catalog might distribute product information via service operations designed for that purpose. Another way of addressing the master data problem is not to distribute it, but to have it accessed from a central service. Every service or other consumer that needs the data uses the central service to access it.

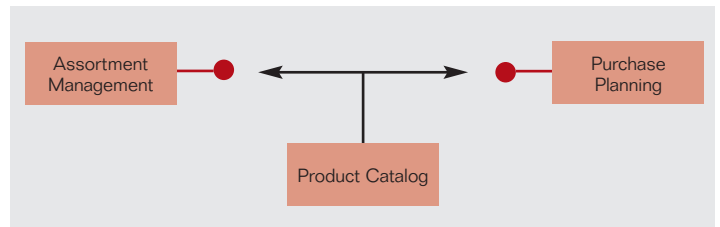


Figure 9: Master Data Distribution

Level 3: Service Enabling Enterprise Applications

Once services have taken hold in an organization such as a large IT department or a commercial software vendor, it is not uncommon for a service enablement program to begin. Usually this happens because the initial projects to create single services or systems of services are quite successful, leading management to want to leverage everything the organization does with services.

For an IT department, a service enablement program may start with an analysis of the services provided by each vendor and then continue with a design program for creating standard services to encapsulate heterogeneous products or services and also designing systems of services to automate unique processes.

For an independent software vendor or commercial software vendor, a service enablement program may involve analyzing many different solutions and distilling patterns for improvement into design contexts and indicators for when they apply.

In either case, the challenge is how to manage the design process and distill the learning about how services should be designed in an organization, define best practices, and then communicate the learning as widely as possible. Design context and indicators are one way of achieving these goals.

This section describes how SAP has used design contexts and indicators to craft a program for service enabling some of its solutions:

- First, a process for using design contexts and indicators during the discovery stage
- Second, a detailed presentation of the design context and indicators for user productivity based on work lists
- Third, a brief description of three additional design contexts that are described in detail at the Enterprise Services Design Center at <http://sdn.sap.com>
- Finally, a discussion of how to create your own design contexts and indicators to capture learning gained in designing enterprise services

Even though executing a program of service enablement may be something that only makes sense at larger organizations, organizations of all sizes can use design contexts and indicators to help in creating enterprise services.

A Service-Enablement Process

Running a service enablement program means equipping a team of solution designers who will analyze many different systems in a process of discovery, designing enterprise services, and then documenting the design of those services. The goal is to find opportunities that may fit into design contexts as well as other opportunities for service enablement.

The discovery stage of the service enablement process involves a sort of walkabout through all of the business scenarios for which services are being considered. The goal is to identify the scenarios that either are in great need of services or that would create even more value if services were added. Two sets of questions can help reveal the right scenarios:

- What are the most important scenarios? What are the pain points? What are the opportunities to create more value? What business drivers might help?
- What business drivers are most important to you? What scenarios might be helped by each business driver?

These two sets of questions interact with each other in a sort of loop.

Solution maps, if you are in an organization that uses them, can be a great way to start this process. If you don't have solution maps, creating them can help understand the needs of a business. The goal is to create a prioritized list of scenarios that are candidates for service discovery and design.

In almost any service enablement program, more opportunities for service enablement will be found than can be implemented. Ideally, the discovery phase of the program would create a prioritized list of opportunities. Then, with a list of scenarios in hand, it is possible to use the procedural model, which specifies a step-by-step process for evaluating the scenarios and designing services to enable them according to any design contexts that are in use.

The Procedural Model for Service Enablement

The procedural model for service enablement is a set of high-level steps through which a group of scenarios can be examined to determine:

- Whether any existing design contexts apply to the scenario
- Whether there are opportunities for service enablement outside of existing design contexts

The procedural model consists of the following steps:

1. A candidate scenario is analyzed to determine which indicators apply to it.
2. If a sufficient number of indicators from a design context apply to a scenario, the design context is selected.
3. Scenarios that match a design context are analyzed. The business processes of the scenario are analyzed along with any existing supporting solutions, and a new group of enterprise services are designed according to the appropriate design context.
4. If a design context does not apply, the scenario is analyzed for service enablement with a single service or system of services. A service or system of services is designed to meet the needs of the scenario.
5. The enterprise services designed are documented in the templates described later.

This process works most efficiently when a scenario fits into a design context and can take advantage of the patterns for service design that are embedded in the design context. But design contexts are a new way of working, and the existing ones cover only a few vitally important areas. In the user productivity design context, for example, the work list–oriented design context is only one of many others, such as key performance indicator (KPI) driven, reporting, or analytics.

When designing a new service or system of services, it is quite possible that a new design context could have been discovered. See section “Creating Your Own Design Context and Indicators” for advice on how to proceed in this case.

General Guidelines for Designing Services and Common Indicators

Once the design context is set, the job of designing services is an iterative process of invention, evaluation, and refinement. The design context provides initial guidance to the general pattern for the services and the way they interact with suggestions for different types of services in each of the functional categories. The previous guidance on how to create services or systems of services should be kept in mind as well.

In all cases, it is important that the design for services be driven from the scenario and business processes. If existing solutions are in place, it will be easy for solution designers to be hemmed in by them and design enterprise services that reflect the structure of the existing solution. This would be a mistake. The needs of the business process must be in the driver’s seat. Of course, once the right enterprise service is defined, it must be implemented, and implementation convenience and reuse of existing functionality sometimes means that the ideal design is compromised. Making such trade-offs properly so that the resulting group of enterprise services really helps extend the range of a solution to fit more requirements is the art of enterprise service design.

The next part of this section will describe the user productivity and centrality design context in detail so that the idea of a design context is made quite clear. Three other design contexts will be described that are available in greater detail at the Enterprise Services Design Center at <http://sdn.sap.com>. The section will conclude with a discussion of how to create a design context and associated indicators.

Service Enablement – Design Context: User Productivity and Centrality

User productivity and centrality is the business driver of Enterprise Services Architecture that will likely have the most design contexts associated with it. This section focuses on a work list–oriented design context that shows how to empower users with self-service user interfaces.

The User Productivity and Centrality Design Context

The user productivity and centrality design context was created to assist those responsible for service enabling mySAP Business Suite solutions, but it applies in general to helping any solution improve its usability. SAP’s main goal in supporting user productivity is to increase the efficiency and effectiveness of people working with SAP’s business solutions, as show in Figure 10.

This includes:

- Increasing process automation to reduce the need for human interaction with the system.
- Increasing service automation (that is, via self-service) to reduce the need for system experts and for additional communication outside the system.
- Changing the way experts work with the system. The goal here is to let the experts steer and guide their business and help the system to automate rather than only working with the system on a transactional basis.

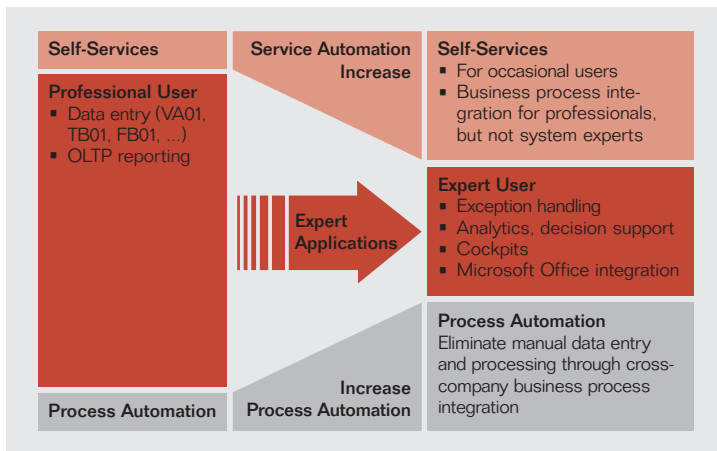


Figure 10: The User Productivity Methodology

The work list-oriented design context applies to users whose work style is driven by work items in a list. There are other work styles in which people are driven more via alerts or by monitors, for example. The overarching goal is always the same. The system should provide the user with all the information and solutions that:

- Help the users to find their work
- Help the users to get their work done as easily as possible by including context information
- Support their decisions

Figure 11 shows a top-down approach to identifying and specifying areas for service enabling for work list-driven users that leads to higher user productivity.

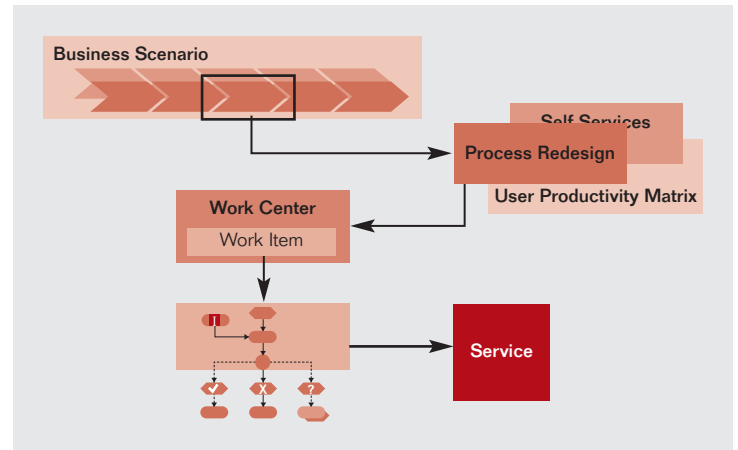


Figure 11: Overview: Applying User Productivity Principles to Find Services

Starting with a business scenario, we look at how people actually work and how the data and functionality of the business applications involved can be brought to the user in the best way (for example, through user interface components in a work center). This leads us to the definition of services needed from the underlying applications.

Before we start, we would like to define a few terms:

- **Task** – A situation in a business process that requires a user to do something. In our case, this usually involves interacting with a business application.

Example: A purchase requisition that needs to be approved by a manager constitutes a task for the manager.
- **Work item** – The representation of a task when it is brought to the user for processing.

Example: The information about the purchase requisition along with the possible actions.
- **Activity** – What the user does when completing a task – in our case, usually what he or she does in the business application.

Example: The activity of the manager approving the purchase requisition includes approving it, rejecting it, requesting more information about it, or routing the request to somebody else.

- **Trigger** – Something that happens in the business that leads to the creation of one or more tasks.

Example: A user creates a purchase requisition that requires approval.

- **Work center** – A collection of user interface elements that allows a user to receive tasks and perform appropriate activities that are common in his or her role.

Example: A part of a work center for a manager may be a list of all purchase requisitions he or she has to act upon along with the possibilities to do so, which in turn might include different user interface elements to see the details, to ask the originator for more information, or to forward the request to somebody else.

Business Process Analysis and Improvement

Figure 12 shows three ways that business processes will be improved in the work list-oriented design context.

Please note that when analyzing the processes, we might come across steps that are manual today but could be automated, which of course would increase the efficiency considerably.

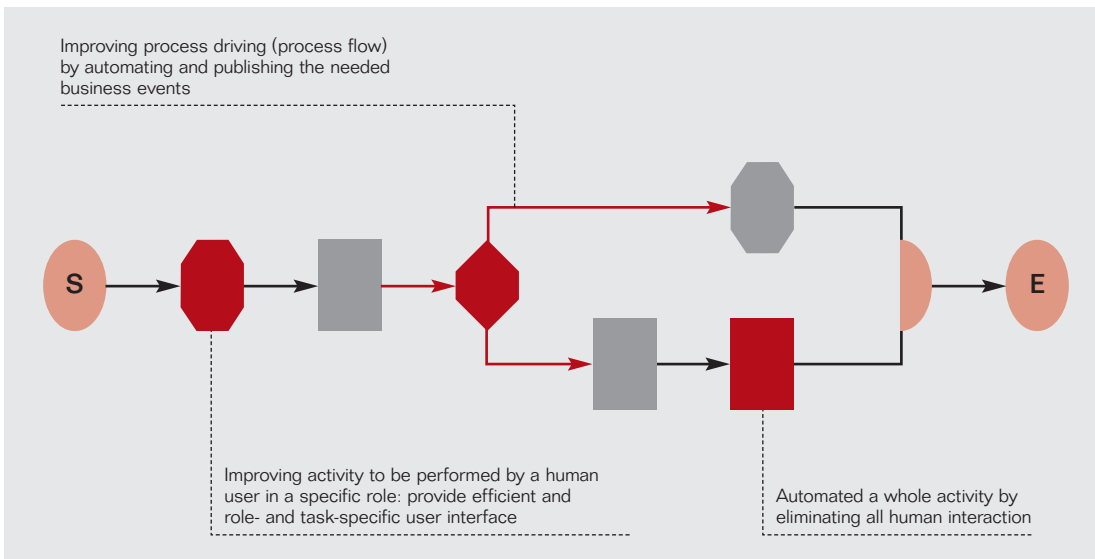


Figure 12: Different Ways of Improving Process Efficiency

User Productivity and Centricity Indicators

The presence of indicators 2, 3, and 4 from the list of indicators shown earlier represents strong evidence that the potential for the creation of a self-service interface exists. Let's reiterate those indicators for convenience.

Indicator 2: Information logically linked and often needed by users is distributed throughout disparate systems.

Indicator 3: Business know-how is decentralized, but the decentral users need support from central users to get their business done. There is a high call or request rate to expert users that results in repeated information retrieval from application systems or the application of well-defined business rules (such as, tasks that could be automated).

Indicator 4: Information is manually transferred from paper into applications or between applications.

In the process redesign, the single process steps are assigned to specific user groups or business roles. It is very important to do business process redesign before assigning process steps to roles, because a task may be shifted from an expert user to a self-service user or even be automated so that a completely different user interface or a background service may be needed. Also, the scope of applications assigned to a role and even the type of applications required may completely change during this redefinition.

Whenever we try to move a task toward self-service, we are likely to create the need for a corresponding expert to handle the task if the user cannot complete the task. If we define self-service components, we therefore also need to define the corresponding expert transactions that can be used by power users, a help desk, or another support organization.

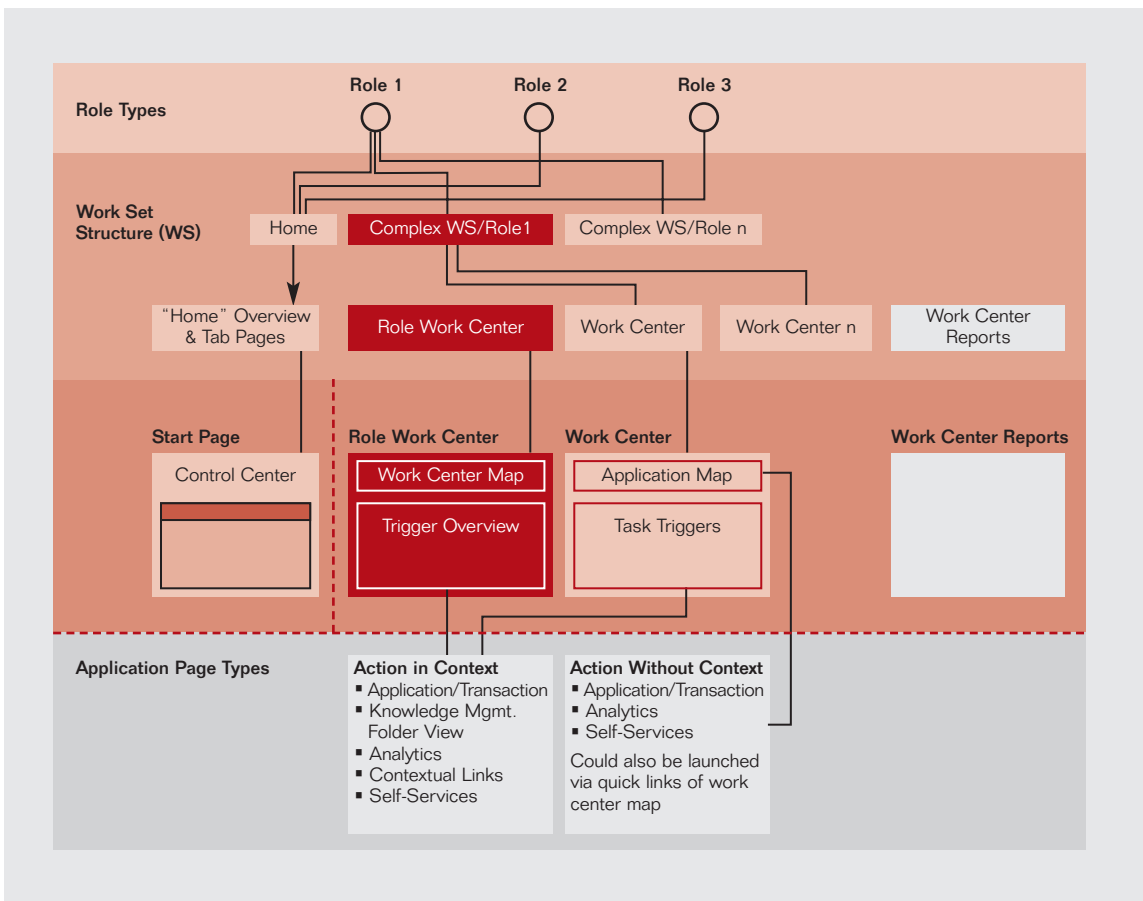


Figure 13: Control and Work Center Concept

The control center and work center illustrated in Figure 13 can be helpful in improving user productivity and usability for self-service users as well as for expert users. The underlying idea is to ask the following questions:

- How does the work get to the user?
- What are the work triggers?

Instead of starting reports to find tasks, tasks should be pushed to the user in his or her work center or there should be tools available to let users quickly monitor their area of responsibility and support their decisions. Different standard user interface elements have been defined for this purpose. The overarching goal of these efforts is to make all the work a user has to do accessible via the control and work centers. Although it might not always be easy to achieve this goal, it is important to do this because it is a prerequisite for the redesign of expert users' applications, as will be shown in the next section.

Service Definition

After preparations have been made, as described in the previous section, the user should find work presented to him or her in his or her work center. Using an event-driven process chain to model and visualize the business process in detail, a work trigger or work item in the work center informs the user about an event and represents a definite status of the business process. Handling this work item means executing a certain task in the business process. Under the condition that the process model is complete, one task can result in only a limited number of well-defined states and trigger certain events.

Even if it is not always possible to identify and specify all possible reactions to a certain event, these considerations should make clear that the system that brings the task to the user “knows” much about the status of the business process, the objects involved,

and the business context in general. In many cases, the possible reactions to the events are limited. This business context can be used to limit the features offered to handle the task to selected functions, methods of the business objects, and information services supporting decision making.

Let's look at a concrete example of a B2B scenario in which the supplier cannot fulfill the purchase order in the desired way. The supplier will send back some information to the purchaser to inform him or her about the changes (for example, possible delivery date, available quantity). When the purchaser gets this information via an appropriate work trigger (a workflow item in work list), the status of the purchase order is well defined and the number of possible reactions is one of the following:

- Accept the changed purchase order
- Reject the changes by either canceling the purchase order and creating a new one to send to another supplier or canceling the purchase order and informing the person who made the initial request
- Start some other activities to get a decision or information to support the decision

Compared to the traditional transaction **change purchase order** that offers all possibilities to change the order, an appropriate service would limit the functionality to the required minimum, offer additional information services that support decision making, or allow the user to start a subsequent process (ad hoc workflow, process to clarify issues, and so on).

Other Design Contexts Related to User Productivity

The design context just presented is focused on work list-driven users. There are also other work styles, such as alert and KPI driven or reporting and analytics. Design contexts for each of these will be developed in the future.

**Service Enablement – Design Context:
Next Business Practices and Process Innovation
(Composite Applications)**

A design context focusing on next business practices and process innovation has been created to help with service enablement when composite applications may be beneficial.

Composite applications are formed by combining elements of existing transactional applications with more collaborative elements such as analytics and other infrastructure components. Composite applications create new business value by reusing, integrating, and orchestrating functionality from existing application assets, with only the selective development of new functionality where needed to fill gaps. They are all about reuse.

From a business perspective, composites automate and enable business processes that are geared toward information workers, knowledge workers, and decision makers as opposed to back-office personnel. In that sense, composites address the last major user community: one that's been largely overlooked by SAP and the enterprise software space over the last 30 years.

To better understand composite applications, let's look at the definition in more detail. First, let's separate the expression and look at each word:

“Application” means:

- Performing new application functionality beyond simply integrating existing systems
- Expanding support to flexible, configurable, collaborative processes beyond the scope of existing applications
- Allowing analysis followed by action, not read-only aggregation of corporate information

“Composite” means:

- Aggregating functionality of existing systems, exposed through Web services
- Crossing traditional application boundaries and user communities
- Creating a comprehensive process and information model

It turns out that indicators 5 and 6, mentioned earlier, are strong evidence that using services to prepare a scenario to be served by a composite application may be the right choice.

**Service Enablement – Design Context:
Business Automation and Process Efficiency**

The design context for business automation and process efficiency is aimed at extending automation through the use of services. Four different sets of indicators show different ways of achieving this goal:

- Indicators 7 and 8 may mean that a supporting process could be enabled by services.
- Indicators 9, 10, and 11 may mean that a service can be created to drive forward processes that previously required manual oversight and initiation. These indicators apply even to processes that cross application or corporate boundaries.
- Indicator 12 may mean that services could help with supporting standards.
- Indicator 13 may mean that processes in several applications could be consolidated into one service.

**Service Enablement – Design Context:
Flexible Deployment**

The design context that focuses on flexible deployment can assist with system consolidation and outsourcing:

- Indicators 14 and 15 may indicate that a service can become a central point for gathering information or a container that allows either centralized or distributed deployment.
- Indicator 16 may mean that a service can encapsulate functionality that could be performed by BPO providers.

Creating Your Own Design Context and Indicators

The Enterprise Services Design Center at <http://sdn.sap.com> offers articles that suggest how to create your own design contexts and indicators. To create a design context requires describing in a general manner how services will improve support for a business process according to a certain pattern.

To be useful, a design context must describe the following aspects of a pattern:

- The nature of the processes that will be supported
- How those processes may be transformed or redesigned by service enablement
- The way in general that services will support the processes
- The benefits of service enablement done in this manner
- Common mistakes or errors to watch out for in this design context
- Indicators that this design context applies to a business process or scenario

Creating a comprehensive system of design contexts and indicators is something that is most likely only going to be performed at the largest organizations. But acting as if each service design project may be turned into a design context can be extremely beneficial. Thinking about the general case helps improve the design of specific services and can encourage brainstorming about how to repurpose and reuse a service in different contexts.

Describing and Specifying Services

The level of detail to which a service must be described varies according to the nature of the service and the context in which it will be used. In general, the larger the organization context and the more people who will be using the service, the more documentation that will be required.

The minimum level of documentation that should be required is enough to make a complete entry in the enterprise services repository, for which the following items must be entered:

- Business objects
- Service interfaces
(with associated operations, message types, and data types)
- Process (ARIS) models
- Business scenario and business process objects
- Mapping objects

Data Group	Columns in Data Group	Description
Service operations	Service operation	Business name of the service operation
	Service operation description	Describing the service operation
	Business case	Clear business case for the service operation
	Technical name	Technical name of the service operation that adheres to any naming conventions in place
	Transaction pattern	Stating what the service operation does – is it request/response, query/response, or notification?
Service provider details	Application/component/engine	Functional business component that will provide the service
	Process group/process	Process within the component that will provide the service
	Implementation notes	Any service operation implementation specific that needs to be taken note mentioned here
	Third party	Putting 'X' if the service operation would be provided by a third party
Planning and availability	Priority	Priority indicated using A-B-C where 'A' is high, 'B' is medium, and 'C' is low priority
	Reuse	Putting the count of scenarios that will consume the service operation – higher the number is better reuse (provide this value only if you are provider of the service operation)
	Effort	Providing person-days needed to provide the service (provide this value only if you are provider of the service operation)

But this is just the beginning of the documentation needed for an enterprise service. At SAP, when an enterprise service is created, a series of spreadsheets are filled out with some of the information shown in the following table. This sort of documentation is quite useful when running a program of service enablement because it allows all of the services being developed to be examined in a uniform way.

In addition to high-level information like this, a document is created to describe the business scenario and how the service and service operations support the scenario. This document has the following sections:

- “Business Scenario Description” (implementation independent):
A description of the business scenario to which the service applies, including such elements as processes and process steps involved in the scenario and the roles that are part of the process.
- “Functional Business Components”: A description of the functional business components that will be involved in the process, how they interact, and the service operations that will be used to support the interaction.
- “User Interaction, Process Efficiency and Automation”:
The service operations for user interface, process efficiency, and automation are described along with how the service operations will be supported by the functional business components.

This overview, while far from comprehensive, will at least provide an outline of the sort of documentation that is needed for an enterprise service.

SAP Developer Network hosts a technical community that connects, collaborates, and contributes to building ESA-based solutions today. For more information about enterprise services, visit SAP Developer Network at sdn.sap.com.

www.sap.com/contactsap

THE BEST-RUN BUSINESSES RUN SAP™



50 076 206 (05/10) Printed in USA.

© 2005 by SAP AG. All rights reserved. SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary. Printed on environmentally friendly paper.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.