Author:
Axel Schuller

Architecture Guideline Series for Composite Applications

Business Logic, Abstraction Layer and Connectivity

Version 1.1

November 2007

Table of Contents:

1 Scope of this document

This part of the 'Architecture Guideline series for Composite Applications' covers mainly the business logic and abstraction layer and backend connectivity of composite applications or short 'composites'. It is done by guiding the reader through the layer, discuss possible options for the layer, and give recommendations for architectural decisions based on experience and best practices acquired in first composite implementation projects.

One benefits most from this part of the guideline series if one has already a general overview of composites as provided in the 'Introduction and Basic Overview' part of this guideline series. It also helps to have a fair amount of knowledge of composite specifications. These specifications are written by either product managers or business experts and explain the composite functionality. For more information on composite specifications, one can review the paper "Guidelines for Specifying Composite Applications" (GSCA). (https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/docs/library/uuid/20844e88-0d01-0010-de9a-eb2d302df7b7)

2 Architecture Overview

Figure 1 provides an overview of the layers of a composite, the technologies to be used in the business logic and abstraction layer, and the communication between the layers.

The following chapters provide detailed information about the technologies used in the portal, process and UI layer. The approach taken is to answer the following questions for each layer:

- What is the layer about?
- What are the technology options?
- Link to 'Guidelines for Specifying Composite Applications' (GSCA) for the relevant high level information

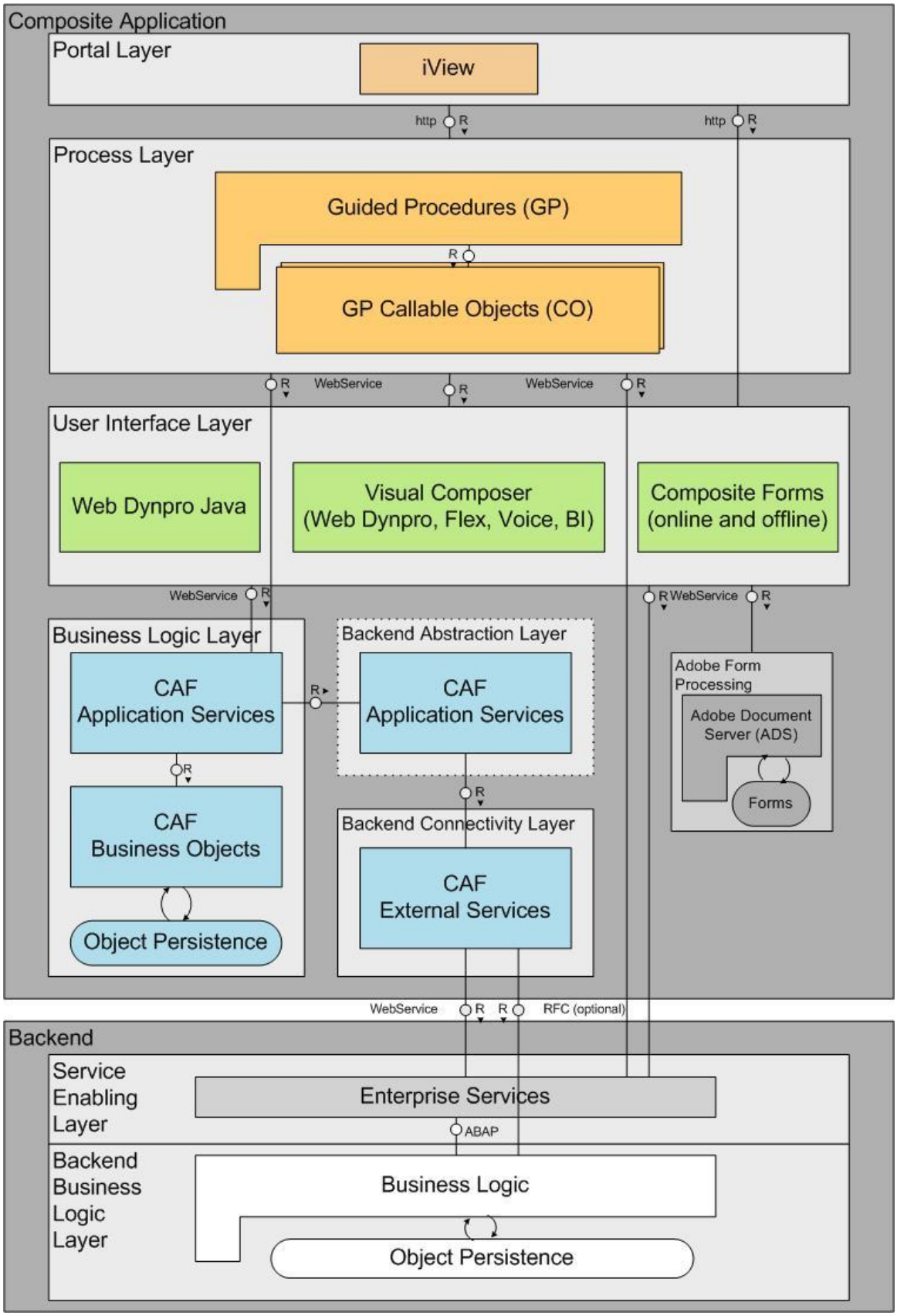

Composite Application
Portal Layer
iView
http
R
http
R
Process Layer
Guided Procedures (GP)
R
GP Callable Objects (CO)
R
WebService
R
WebService
R
User Interface Layer
Web Dynpro Java
Visual Composer
(Web Dynpro, Flex, Voice, BI)
Composite Forms
(online and offline)
WebService
R
R
WebService
R
Business Logic Layer
CAF
Application Services
R
CAF
Business Objects
Object Persistence
Backend Abstraction Layer
CAF
Application Services
R
Backend Connectivity Layer
CAF
External Services
Adobe Form
Processing
Adobe Document
Server (ADS)
Forms
WebService
R
R
RFC (optional)
Backend
Service
Enabling
Layer
Enterprise Services
ABAP
Backend
Business
Logic
Layer
Business Logic
Object Persistence

Figure 1

3 Business Logic and Abstraction Layer

Within the Business Logic Layer the business objects (BOs) specific to the composite are implemented, see GSCA chapter 4.8 'Business Objects Overview'. BOs encapsulate reusable functionality and may expose services. These services can be either local or remote if published for reuse by others.

An Abstraction Layer is not required but recommended in order to keep higher layers independent from service implementation details and making them more flexible. If the services used are configurable it is especially useful to have an Abstraction Layer to be able to deal with the configuration rather than to have to deal with them in the Business Logic Layer. Technically there is no difference between the Business Logic Layer and Abstraction Layer, it is just a question of design.

From a model driven aspect, SAP CAF should be used to model the business objects (attributes, nodes and services) and generate and manage the local and remote persistency of the business objects. This helps to reduce manual coding and to get faster and safer results.

3.1 SAP CAF

SAP CAF provides the ability to import services provided by the backend systems as external services, model business objects as entity services and implement business logic in Java as application services.

Main Advantages:

- Modelling of Business Objects and generation of the persistency, CRUD- and optionally Query methods for this model.
- Modelling of services incl. the data types needed for their signatures
- Exposing the logic as (web) services to the user interface and/or process layer
- Authorization concept for Business Objects and services

3.1.1 Business objects

A Business Object (BO) in SAP CAF is essentially a data persistency model, defined by a set of attributes and relationships to other Business Objects, the lifecycle methods Create, Read, Update, Delete (CRUD), each working on a single Business Object instance per call, and query methods to retrieve multiple instances based on search conditions.

Regarding persistency, there are two options: remote and local:

- Remote persistency means the actual Business Object instances reside in an external system, which has to be accessed by means of so-called External Services mapped to the CRUD and query methods of the SAP CAF BO (hence, such SAP CAF BOs are called "Remote Business Objects"). SAP CAF offers the possibility to either connect to RFCs or to document-style Web Services via External Services.
- Local persistency means SAP CAF will create an own database model for you that reflects the structure of your BO, which is therefore called "Local Business Object" (e.g. simple attributes with cardinality 0..1 or 1..1 will be reflected by database table columns, whereas cardinality 0..n or 1..n, as well as a reference to another Business Object will lead to an additional table linked via foreign key relationship).

Moreover, there is a special persistency type "Custom", which basically allows you to redefine the out-of-the-box CRUD methods, i.e. you can implement your own persistency logic that way.

When creating Business Objects in SAP CAF, please keep in mind then following suggestions:

✗ **Don't use Remote Business Objects**

Remote Business Objects have the following disadvantages:

- only query and CRUD operations with quite strict signatures allowed, which makes mapping to complex signatures of Enterprise Services difficult/impossible

- local persistency footprint for each data record read from the remote/backend system via the mapped service, which means sub-optimal performance
- at this time, there seems to be no clear benefit over an application service operation mapped to an External Service, therefore they are a somewhat redundant concept.

☑ **Only use local Business Objects to store data existing in your Composite**

This means: if you have to persist some data specific to your composite, CAF local BOs are to be used for that.

☑ **Handle potential lock conflicts with local Business Objects.**

If there is an optimistic lock conflict (which will occur when you try to update a BO based on 'dirty read' data, i.e. by a data record older than the most currently committed version), CAF will throw a CAFOptimisticLockException. In this case, you'll typically have to notice the user that the data submitted by him is based on an outdated state of the respective BO and present him the current state, so he can merge his entered data with that.

In case a pessimistic lock occurs (which can only be the case if a pessimistic lock for a BO instance has been requested pro-actively, i.e. there is no inherent "automatism" like for optimistic locks), the framework will throw a CAFPessmisticLockException. In this case, you can only presently a message to the user that the respective object is locked at the moment and he has to re-try later. In general

- o pessimistic locks should be set/tested before data entry by a user.
- o pessimistic locks should only be set if it's clear that optimistic locks won't be sufficient for data integrity, which should rarely be the case.

See the chapter 'Transaction Handling' in this guide for more details on optimistic locking.

Also notice that optimistic locking only works with authenticated users, i.e. anonymous requests will never notice lock conflicts.

✗ **Do not create an own implementation of your Business Object. In particular, don't implement own persistency logic.**

It is possible to overwrite the generated CRUD methods of a Business Object.

However, it is recommended to keep all custom coding in Application Services because of consistency reasons and to reduce potential error sources (like forgetting optimistic lock conflict checks, permission checks, tracing CRUD method calls, …). Moreover, it would reduce the model-driven character of our Business Logic Layer in general.

✗ **Do not directly expose Business Object operations to the upper layers of your composite.**

Main reasons:

- Fixed/inflexible operation signatures, partly not complying with our needs/requirements towards service consumption (e.g. you cannot add a result sub-structure adhering to GDT Log as needed by our error handling concept).
- No out-of-the-box possibility to restrict result size for query operations (would only be possible by creating a custom BO implementation, but we want to refrain from that, cf. last point). Since SAP CAF also doesn't support some kind of paging mechanism (because of the statelessness of its services), this may lead to huge data sets transported to the UI at once.

✗ **Avoid pure external data replication**

Replication of persistent data between a Composite Application and its Backend systems adds complexity since the (semantically) same instance of a business object can be changed independently at several places, thus risking out-of-sync situations, i.e. depending on the respective scenario more or less severe inconsistencies.

Ways to avoid the need for data replication:

- a process design creating the Business Objects in the Backend system not before the process within the Composite is finished

- Storing only references/keys to the Business Objects in the Backend Systems and not persisting redundantly their data in the Composite

Nevertheless there may be cases where data replication is sensible:

- Scenarios where you'd otherwise would need to perform a huge number of backend accesses or you would have to transport a large amount of data again and again (remember SAP CAF is stateless, so keeping this data in session context is not possible), leading to unacceptable performance/scalability
- Collaboration Objects, which are a representation of a backend object, but which shall not be automatically updated with every change within the backend (i.e. where out-of-sync situations are uncritical).

Generally, the following points apply to data replication:

- In case of rather short-living but high-amount transactional data (e.g. purchase orders), replication is most problematic since frequent re-syncs are not feasible on the one hand but out-of-sync situations are much more likely on the other. Therefore, try to avoid data replication by all means in this case.
- For long-living master data, out-of-sync situations are not that likely, so replication might be allowed and designed on a case-by-case basis.

3.1.2 Application Services

Application services allow for custom programming, e.g. to transform/combine data from local Business Objects, External services and other Application services or performing any kind of custom business logic.

☑ **Use Application Service operations to implement the business logic which is specific to the composite**

☑ **Adhere to the SUN Java code conventions for your custom coding which can be found under http://java.sun.com/docs/codeconv/**

When working with application services, the following rules apply:

☑ **Put simple business logic directly into an Application service operation**

Under the "Implementation" tabstrip of your Application Service, click the <Application service name>BeanImpl.java link to create a class deriving from an abstract generated class containing the CAF specific parts. In the created class, you can override the methods corresponding to the operations of your Application service then.

☑ **If business logic gets more complicated, distribute (parts of) it to additional methods and/or classes to get a clearer design**

Just create the needed additional classes/packages under the src folder of the ejbmodule DC created for your SAP CAF project. They will be added to the build-result of the corresponding ear project on the next build then.

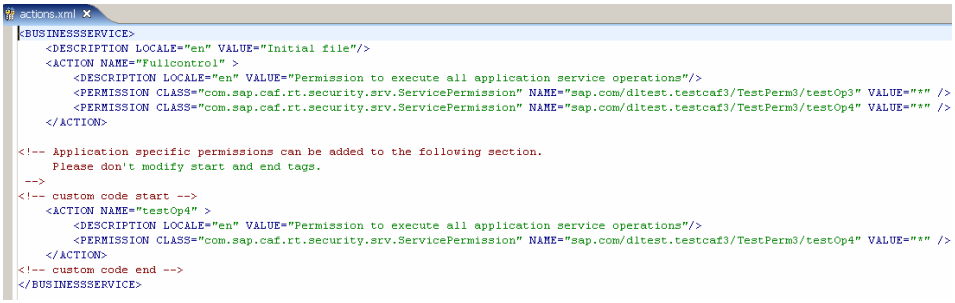
☑ **Clarify with your product owner which operations need authorization checks**

☑ **In case there are such operations: make sure authorizations can be configured per operation**

To enable an administrator to assign different authorized roles to different App. Service operations in a modification-free way, you have to do the following:

- In the permission DC of your SAP CAF project, you can find a file named "actions.xml" located in folder /src. By default, this file contains one action entry "fullcontrol" having one permission per Application Service operation with permission checks enabled.
- After deploying the permission DC's SDA file [you have to do that explicitly, i.e. it won't be automatically deployed from Composite Application perspective with your CAF project at the time being], you can find the mentioned "fullcontrol" action in the User Management tool. When assigning this action to a UME role, each user assigned to that role will be allowed to call all operations of your App. Service. This is typically undesirable.

- Therefore, you should add an action per operation to the “custom code” section of the actions.xml. Use the “fullcontrol” action as a template and name your action exactly like the operation it is referring to:



After re-building and re-deploying your permission DC, you should be able to assign the new action to an UME role in the User Management tool, allowing the respective users only to execute the one referred operation.

☑ **Enable all Application services as Web Service if they are to be accessed by Process Logic and/or UI layer.**

In principle, there are the following options to consume an Application service operation from UI and/or process layer:

- Enable the Application service as Web Service. This can be achieved by choosing “Expose service as Web Service” from the Application service context menu in the CAF Composite Application Explorer view, which will start a wizard where you can add all (applicable) operations to the Web Service interface.
- JNDI lookup & direct call of EJB interface (since each Application Service is an EJB Session Bean underneath).

☑ **Application services operations should access other Application services operations by EJB method calls (not through web service interface)**

☑ **Provide all info, warning and error messages in a sub-structure of your Application Service operation output parameter, adhering to the GDT ‘Log’,**

Re-use the Log structure type generated when importing an Enterprise Service as External Service (typically, all Enterprise Services use the GDT Log in their signatures).

Temporarily import some Enterprise Service as External Service to get this Log type, even if you actually don’t need to connect to a backend service (delete the External Service and all unnecessary generated types again in such (probably rare) case).

✘ **Don’t throw exceptions in Application Service operations exposed as Web Service operations. Use GDT ‘Log’ with SeverityCode 3 (Error) or 4 (Abort) instead.**

Note that exceptions would be communicated as SOAP Fault Messages, which cannot be handled as desired by us in some cases (for example, you can only react with Process Exceptions to SOAP Fault Messages in GP, cf. chapter “Technology Selection for Process Layer”).

Moreover, CAF only supports passing a string as payload of SOAP Fault Messages for App. Service-based Web Services, whereas GDT Log also allows for passing additional data (e.g. WebAddress pointing to some detail information on the error).

☑ **Only add operations of type ‘CUSTOM’ with proper signature to Web Service interface. Cut these operations in a coarse granular manner according to the needs of your composite application.**

“Proper” means you should try to stick to Enterprise Service guidelines as close as possible. Especially, try to re-use (parts of) those types created when importing Enterprise Services as External Services.

“Coarse granular” design means to avoid distributing logic that should belong to one transaction over several operations if possible. Note that all accesses to Business Objects performed from one

Application Service operation can be executed in the context of one JEE transaction (and hence, be rolled back if necessary).

Enterprise Services are built for reuse per Business Object applying standardized patterns and not for a single consumer only. In contrast to that the Application Service Operations of the composite are built dedicated for its User Interface or Process Logic only. So you should take the freedom to cut those according to the needs of your applications potentially crossing BO boundaries.

☑ **Make use of logging and tracing**

The topic *How to Write Useful Log and Trace Messages* in the SAP Help Library gives an overview of what should be logged and traced and how to achieve this with the SAP Logging API. Summing it up:

- Traces are primarily targeted towards developers, whereas logs are targeted towards administrators.
- Log messages are to be emitted via class com.sap.tc.logging.Category (since Categories describe the origin of messages from a more semantical point of view, e.g. '/*System*/*Database*'), trace messages are to be emitted via com.sap.tc.logging.Location (since Locations describe the origin of messages from a more technical point of view, like e.g. a fully qualified Java class name).
- The emission methods of com.sap.tc.logging.Category all take an additional input parameter 'loc' of type com.sap.tc.logging.Location ➔ log messages are also traced at the same time, given that their priority is not below the threshold of the passed Location (which is 'Warning' by default, so e.g. info messages will not go to the trace).
- Categories as well as Locations are maintained (but not created) in NW Administrator under 'Problem Management' → 'Logs and Traces' → 'Log Configuration'. Both are organized in hierarchies, inheriting settings (especially destination and maximum sizes) from their respective parent nodes.
- By using the methods Category.getCategory or Location.getLocation (overloaded with various signatures), you can instantiate existing Categories/Locations or create new ones ad-hoc implicitly (if the specified one doesn't exist yet).
- Rules for Category and Location selection:

  ☑ **Use Location.getLocation(<Current class name>.class) to instantiate a tracing location**

  This will use the fully qualified name of the given class to retrieve or create a location hierarchy along the package path as needed, giving you the leaf location.

  ☑ **Use Category.getCategory(Category.APPLICATIONS, <composite name abbreviation>) to instantiate a logging category**

  The new category will hence be created under root category 'APPLICATIONS', which is present by default after SAP NetWeaver CE installation.

3.1.3 External Services

☑ **An External Service is to be mapped to one Web Service Destination created in the SAP NetWeaver Administrator**

See *Configuring External Services in CAF Runtime* the SAP Help Library for how to do this destination mapping.

If an External Service call should be mapped to multiple endpoints, this has to be done by means of XI on customer side.

Possible scenario:

A customer desires one of our composites to read and combine data for business object "Supplier" both from an ERP and a SRM system.

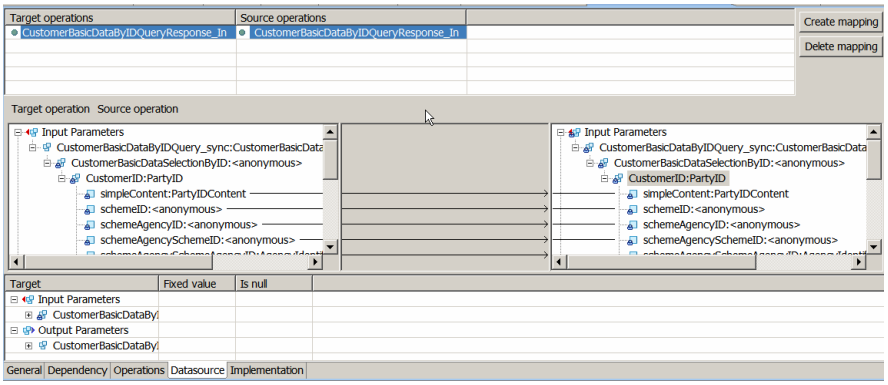
By default, our composite would do only one service request to get the supplier data (from an ERP system typically) however. So the customer can modify our composite, extend the backend abstraction layer by a custom connectivity DC or use XI to split the original request into two new ones as well as combining their results, where the latter one is maybe the “most elegant” solution if he already has an XI system in his landscape.

☑ **External Services may only be accessed out of Application Services using mapped operations**

☑ **Re-use types generated on External Service import when defining Application Service operations to be mapped.**

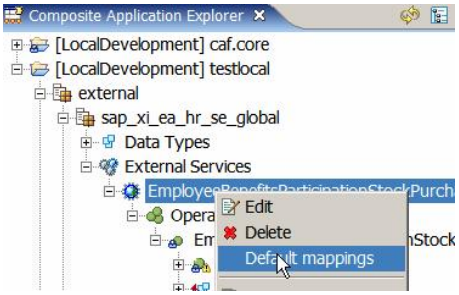
Note that programmatic calls of External Services are not supported in the SAP NetWeaver CE anymore (i.e. there is no proxy class generated for an External Service). Instead, an operation of an External Service can only be accessed through a mapped Application Service operation.

To create such mapping, uncheck the “Implemented” checkbox of the Application Service operation and switch to the “Datasource” tabstrip to assign the desired External Service operation. By clicking this assignment, you’ll get a view where you can perform the actual parameter mapping.



Note you can re-use types CAF design time generates on External Service import to define Application Service operations. This will facilitate the mapping to the respective External Service in many cases.

It is also possible to create fully mapped application service operations ad-hoc from an External Service, hence giving you an equivalent to a type proxy.



✘ **There is no standard way for “pushing” data to UI layer as of today. Don’t try to implement one on your own.**

This means new/updated data has to be requested by an UI from the Business Logic Layer pro-actively. Pushing data to the application UI, which means changing displayed data without explicit (i.e. user trigger) or implicit (e.g. by means of WD TimedTrigger Control) UI event is not feasible at the time being. You should also refrain from investing into a custom implementation of such push mechanism, e.g. by “service-enabling” your UI layer and using such service via an External Service.

4 Backend Connectivity

It is recommended to check the GSCA list of required and available services and consider the following:

- ☑ **Synchronous Enterprise Services consumed via Web Service Protocol should be used for backend connectivity to be fully eSOA compliant**
- ✘ **No asynchronous communication between a composite and a SAP NetWeaver 7.0-based backend system (e.g. SAP ERP 2005) is currently supported**
- ☑ **Web Services are recommended but BAPIs and RFCs may be used**
- ☑ **The Composite only acts as client calling the Enterprise Services**
- ☑ **Enterprise Services services are stateless and atomic (see Transaction Handling in this guide)**
- ☑ **No composite-related upgrades of the backend systems should be required.**
- ✘ **No outbound calls or events from backend to composite are currently supported**

Ideally a composite is not 'aware' of a specific backend since it communicates via the backend connectivity layer. This layer ensures that the composite can 'talk' to the service enablement layer and with the right service provider instances.

The optional SAP NetWeaver Exchange Infrastructure (SAP NetWeaver XI) may be used as the messaging middleware for service communication, connectivity, transformation and portability to connect the composite with its backends. Due to a dedicated technical (background) user used in SAP NetWeaver XI during message processing, direct usage of SAP NetWeaver XI as connectivity layer is not recommended.

In the target Enterprise SOA based architecture, the technology of choice for connecting composites with the backend is Web Services. If this requires an unacceptably high level of effort, BAPIs or RFCs can be used instead, but since it is not compliant with enterprise SOA it is not recommended

Backend independency keeps the composites flexible and reduces the effort otherwise needed to adapt to different backends. For this reason composites should be loosely coupled to the components on which they are based and act only as clients of enterprise SOA services. Shared volatile states and transactional locks between composites and their underlying components do not comply and would not allow stateless and atomic communication. Asynchronous communication is especially needed, to decouple the composite from the backend in cases where the composite is not dependent on the direct result of the service call, e.g. when Updating a Business Object. By this the following advantages could be achieved:

- Performance by non-blocking execution of the service
- Reliable Transportation of several messages (in order)

Events sent from the backend to the composite would be especially needed in cases, where the backend wants to inform the composite about exceptional situations, so that the composite is able to handle those. As of today no coherent event infrastructure exists, which allows to provide events within the backend and consume them in the composite.

- Ability to send and queue messages, if the receiver is not available

A Composite is reusing the functionality and data provided by backend systems. Within enterprise SOA this is done by consuming Enterprise Services. An Enterprise Service is defined as a service which provides business functionality and which is published by SAP in the "SAP Enterprise Service Inventory" (currently ES Workplace). Enterprise Services are structured according to a harmonized enterprise model based on process components, business objects and global data types. They are well documented, guarantee quality and stability and are based on open standards. The consistent definition of Enterprise Services is ensured by usage of Patterns. The most important patterns for the usage within a composite are for example Manage Business Object, containing operations to Create, Read and Update a Business Object Instance.

Besides Enterprise Services also services provided over the internet could be consumed by a composite. By the usage of those a Composite can become a MashUp, which is defined as a website or web application that seamlessly combines content from more than one source into an integrated experience.

4.1 Backend Connectivity Configuration

The configuration concept is based on the idea to use logical destinations within all web service consuming tools to define the backend to call. Logical destinations are used as abstraction layer above the physical endpoints. Within a configuration it is possible to assign physical destinations for the landscape to these logical destinations.

- ☑ **Logical Destinations have to be used for all consumed Enterprise Services and BAPIs/RFCs**.

To support reuse composites should integrate into existing system landscapes without the requirement of composite-related upgrades of the backend systems.

A composite has to be deployable independently of other composites. Reuse between composites is only allowed within dedicated reuse of software components.

5 Transaction Handling

☑ **eSOA services are stateless and atomic**

Stateless means that no state is kept in the backend between two service calls.
Atomic means that each service updating the database does its own commit.

As a consequence it is not possible to combine several service calls into one Logical Unit of Work, which is ended by a Commit or Rollback nor is it possible to keep locks in the Backend between the time the data has been read and when it is updated.

The change operation overwrites, it does not check whether somebody else has changed the same Backend Business Object instance since the last read. This means that if somebody else has changed the same instance those changes will be overwritten without any warning.

The update operation checks for concurrent updates. It checks whether somebody else has changed the same Backend Business Object instance since the last read based on a version ID passed by the Read service. This means that if somebody else has changed the same instance an error message will be sent back. It is then up to the composite to react on this:

- Ask the user and call the change operation
- Cancel the changes (data entered by user is lost)
- Offer a screen to merge the changes.

These update operations are only implemented when really needed. This means in cases with a high probability of concurrent updates, or in cases where the “last-one-wins” logic of the change operation is not acceptable.

This additional checking service is especially important for interactive forms with a check and a submit button and in GP with separated UI and Background Callable Objects.

6 Glossary

Term	Definition
Action	An object that connects to the application or service called by a process step. Actions link the process context to the application context. They are configurable and reusable. The resources and results of an action are defined by their input and output parameters. These parameters are assigned to the corresponding application parameters. Actions can call the following applications: WebDynpro components; iViews; interactive forms.
Action Gallery	Gallery of reusable, generalized actions from which a composite process can be built.
Application	A collection of business processes required to address specific business needs, implemented via a set of application and software components running on a platform.
Application Component	From the software development point of view an application component is constructed of software components. From the runtime instance point of view, common persistency and transactional behavior are guaranteed in an application component instance. From the customer point of view, the application component forms the unit of operation and running business functionality.
Application Platform	A software platform that provides a foundation of pre-integrated components, processes, engines, and business objects covering aspects of various business areas, supporting business process management of core processes, and enabling their reuse. SAP's Application Platform is an integral part of the Business Process Platform.
BAPI	Business Application Programming Interface: Standardized programming interface that allows external access to data in SAP systems. Implemented as a method of a business object type in the Business Object Repository (BOR).
BPM	Abbreviation for Business Process Management
Business Configuration	Business Configuration is the adaptation of software to specific business needs in a easy and effective way
Business Object	A representation of a uniquely identifiable business entity described by a structural model, an internal process model, and one or more service interfaces. Business processes operate on business objects.
Business Object Node	A semantically-related set of attributes of a business object. Usually business object nodes provide core services. Example for business object: sales order, sales order header and sales order item are business object nodes.
Business Object Repository (BOR)	SAP application developers or customers can store object type descriptions of their business objects (e.g. customers, material, orders, and service messages) in the BOR. The characteristics of an object (material master) become attributes (e.g. material group, material number, industry sector). The methods determine what can be done with an object (e.g. create, display, prepare to delete) and the events provide information about the status of an object (created, changed, and so on).
Business Process	A business process is a set of activities transforming a defined business input into a defined business outcome.

Business Process Management	Business Process Management describes the modeling and analysis of business scenarios and business processes, orchestration, automation and deployment of these scenarios, and active monitoring of the performance of these business scenarios. BPM is an integral part of eSOA and will be the basis for composition of new services and many composite applications
Business Process Platform (BPP)	The combination of SAP's Application Platform with SAP's technology platform. It supports the creation, enhancement, and seamless execution of business processes and business scenarios.
Business Scenario	A business scenario is a sequence of business processes to achieve key business objectives. A business scenario is either specific to one industry (in which case it is called industry-specific scenario) or could be applicable to multiple industries (in which case it is called cross-industry scenario).
Business Process Step	A task or an interaction performed by a process component either with or without human interaction and together with other steps forming a business process.
Component	A modular piece of software, offering services accessible via interfaces.
Composite	An application that uses data and functionality provided as services by platforms and applications and combines these into coherent business scenarios with user-centric processes and views, possibly supported by own business logic and specific user interfaces.
Composite Application	synonym Composite
Composite Process	Assembles a business process by re-using application components, from possibly disparate systems, into a Single Orchestrated UI experience.
Composite View	Single summary page that incorporates multi-source content to provide a comprehensive view of the subject.
Composite Application Framework (CAF)	The Composite Application Framework (CAF) provides the tools, methodology and environment for building and running composites. It enables model-driven and template-based access to all levels of composite application development, while leveraging SAP NetWeaver capabilities.
Control Center	The Control Center is the user's home area in SAP NetWeaver. It consists of a set of pages organizing the user's activities across, and beyond roles. It summarizes and combines all information the user needs for structuring, organizing, and monitoring daily work.
Core Service	An elementary service based on standard sets of interfaces which completely encapsulates and controls the state and behavior of a business object node thus provides direct access to the business object. Example: Create, update, delete, or query services for sales order item or sales order header.
Compound Service	An application service, using two or more core services to operate on multiple nodes of one or more related business objects. Is usually used for message-based process integration.
Enterprise Service	A compound service used in the execution of business processes, having a significant meaning and impact for the business of the enterprise, fulfilling strict standards regarding version compatibility and stability, and built on Web service technology.
Enterprise Service Oriented Architecture (eSOA)	SAP's blueprint of a service-oriented architecture.

Enterprise Service Infrastructure (ESI)	SAP NetWeaver's environment for defining, developing, identifying, invoking and managing services according to the Enterprise Services Architecture.
Enterprise Service Inventory	The list of Enterprise Services and associated definitions that are being published by SAP in order to help SAP's partners and customers plan their own Enterprise Services Architecture road map.
Enterprise Service Repository (ESR)	The central repository in SAP NetWeaver where Enterprise Services, Business objects and Business Processes are modeled and their metadata is stored. This repository is an integral part of the Enterprise Services Infrastructure.
Floor Plan	A floor plan defines the composition of user interface building blocks on the screen. Depending on the floor plan, only certain user interface building blocks and sequences are allowed.
Guided Procedure	User interface and tool environment for modeling composite processes.
GSCA	Abbreviation for 'Guidelines for Specifying Composite Applications'
Interface	The procedures, codes, and protocols that enable two entities to interact for a meaningful exchange of information.
Interface Pattern	An interface pattern describes standardized interfaces with a standardized behavior.
iView	Program that retrieves data from content sources located inside or outside an organization via a Web protocol and displays it in the SAP Enterprise Portal content area.
Logical Deployment Unit (LDU)	A Logical Deployment Unit is a piece of software that can be operated on a separate system isolated from other pieces of software.
Message	A message is information conveyed from one instance to another with the expectation that activity will ensue.
Model	An isomorphic description of a physical, abstract, or hypothetical aspect of reality which serves to communicate, to construct, and analyze that aspect. Within software engineering models are created, on the one hand, to represent reality in a way that supports the construction of software, and, on the other hand, to act as abstract descriptions of certain aspects of software systems. In certain cases, models can be used to generate software or be interpreted directly at runtime.
(Service) Operation	An operation is the abstract description of a method with a set of messages assigned as signature. A service interface consists of a set of operations.
Object Work List	The Object Work List (OWL) is a list of business object instances which are displayed in the content area of a Work Center. The set of instances is relevant to the current activity. Each item in the list contains a link to various details about that business object instance. An example is: Open Purchase Order Requests.
Pattern	A description of a standard solution to a common problem in software engineering and design. Example: User interface pattern, design pattern, program interface pattern
Process Agent	Process agents are system components, which are called by the event handler to trigger subsequent process steps. Based on the object's status, process agents may launch local workflows or initiate cross-component and B2B communication by sending messages via well-defined outbound services.

Process Component	A modular, context independent, reusable piece of software which exposes its functionality as services. Usually a process component realizes one business process. A business object is assigned to exactly one process component.
Process Component Interaction Model	A Process Component Interaction Model represents the semantically closed set of interactions between two Process Components. The involved business objects, process agents, interfaces, operations and messages are shown.
Process Model	Process Model is the abstract description of possible business process flows.
Role	The collection of activities that a person performs to participate in one or more business scenarios
Service	A task or an interaction performed by a provider upon request by a consumer.
Service Enabled	Self-contained functionality accessible via the Web Services standard.
Service Interface	An interface by which a provider offers functions to a service consumer.
Service-Oriented Architecture (SOA)	Service-Oriented Architecture is the organization of a system in a way that components can be invoked and interface descriptions can be published and discovered.
(Service) Operation	See operation
Software Component	A software component is composed of development components and is the building block for one or more application components. It is the unit of versioning, upgrade and installation.
Software component version	Release of a software component. Building block for application components. Example: SAP KERNEL 6.40 32Bit
Technical Transactions Model	The Technical Transactions Model describes which patterns applications have to follow to get all involved business data consistently persisted. For business objects within the same Logical Deployment Unit (LDU) this is achieved via database integration whereas for business objects from different LDUs it is achieved by message-based integration.
Universal Worklist (UWL)	The Universal Worklist enables SAP portal users to combine work items from different systems in the Business Workplace on a single interface.
UWL	Abbreviation for Universal Worklist
Web Service	A Web Service is a software system identified by a URI [RFC 2396], whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.
Work Center	A Work Center organizes data and activities of a specific task domain (for example, purchase order management, invoice management), and is a logical unit of semantic user interaction, administration, authorization management, packaging, and deployment. It consists of a set of pages organizing and supporting the user's activities in this area (workset).
Workflow	Workflow processes are a specific type of business process which typically requires user interaction. Typical use cases for workflow are process automation, assignment of tasks to users and adding new process steps.

7 Copyright