Author:
Axel Schuller

Architecture Guideline series for Composite Applications

Introduction and Basic Overview

Version 1.0

October 2007

Table of Contents:

1 Scope of the series....4

2 Architecture Goals....5

2.1 Reuse of Services....5

2.2 Model-Driven Development....6

2.3 Enterprise SOA Compliant Architecture....6

2.4 User orientation....6

2.5 Short-time-to-Value....7

2.6 Adaptability....7

3 Basic Architecture of a Composite....8

3.1 Service Enablement Layer....9

3.2 Backend Connectivity Layer....9

3.3 Business Logic Layer....10

3.4 User Interface Layer....10

3.5 Process Layer....11

3.6 Portal Layer....11

4 Copyright....12

Acknowledgement

The contributions of the Industry Composite Development team made the writing of this document possible

1 Scope of the series

The SAP NetWeaver Composition Environment (SAP NetWeaver CE) provides a lean, service oriented and standards-based solution. An easy-to-use environment to model and run composite applications, SAP NetWeaver CE accelerates business process innovation and leverages existing IT-investments by simplifying the integration of heterogeneous environments and legacy systems into user-centric applications.

Composite applications or short ‘composites’ reuse existing services and data to initiate new business practices and provide a view of processes and data, facilitating user interaction and collaboration. They act as mediators between users and user-centric processes on one side and data and business processes on the other. The services and data can be provided by SAP’s Business Process Platform, SAP ERP as well as by service-enabled third-party solutions (service-enabled = self-contained functionality accessible via the Web Services standard).

SAP NetWeaver CE provides a toolset and runtime for developing, running, and managing efficiently composites using SAP's enterprise SOA. With this new breed of application, architects responsible for the technological architecture of a composite are faced with the challenge of choosing the right technologies for the respective layers of such a solution and their optimal interplay. The layers making up a composite are:

With this new breed of application, architects are faced with the challenge of choosing the best technologies for the layers of a Composite and their interplay. The layers making up a composite are:

- Backend Layer
- Business Logic and Abstraction Layer
- User Interface Layer
- Process Layer
- Portal Layer

This architecture guideline series aims to explain how to approach composites from an architect’s perspective. It is done by guiding the reader through each layer, discuss possible options for each layer, and give recommendations for architectural decisions based on experience and best practices acquired in first composite implementation projects.

It is not the intention of this guideline series to go into technical details about how to implement certain functionalities or describe functions and features of particular tools. It is solely focused on the architectural concepts of a composite application, especially the layering of an application and where to implement which logic. Additionally it helps the architect to understand how to translate given composite specifications delivered by product managers or business experts into software architecture that meets all required functions and features. Discussions and how-to-guidelines about implementation details can be found in the NetWeaver Developer Guide. Although these technical details are omitted, it should be clear that the intention is to help with architecture decisions for composites being primarily built with SAP’s NetWeaver Composition Environment. Therefore the guideline series covers Guided Procedures for process modeling, SAP NetWeaver Visual Composer, Web Dynpro and SAP Composite Forms by Adobe for UI modeling, and SAP Composite Application Framework for business objects and business services modeling.

One benefits most from this guideline series if one is already familiar with the above mentioned frameworks and technologies. 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) which details in a less technical manner, how to specify all composite relevant aspects for the architect (e.g. process flow, roles, user interfaces, business objects, and services).

2 Architecture Goals

The main composite architecture goals are to support efficient development of new applications which can easily be adapted, allow flexible backend connectivity and reduce maintenance efforts. These goals can be achieved by the following high-level guiding principles:

- Optimization of development efficiency by the **reuse of services** and **model-driven development**
- Increase of flexibility and backend independency by an **enterprise SOA-compliant architecture**
- Improved user acceptance by an **user oriented approach**
- Combination of existing services to new processes, extending a composite by a model-driven approach, and simple installation and configuration contribute to a **short-time-to-value**.
- Composites typically face the following challenges:
    - Different landscapes
    - Business Logic needs to be independent of backend systems and UIs
    - UIs and Processes are likely to be needed slightly different than originally designed

  Taking these challenges into account an easy **adaptability** is very important for a composite application.

Each of those principles will be discussed in greater detail in the sections to come.

2.1 Reuse of Services

An essential part of the composite idea is the reuse of available services to implement new business processes.

To support reuse, composites must integrate into existing system landscapes without the backend systems requiring composite related upgrades. This non-invasiveness is one of the key features of composites. 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 backend services on which they are based, resulting in a new logical application tier with its own lifecycle.

The advantage of a composite having its own lifecycle is that the delivery of new versions is independent of the release cycle of the underlying backend systems so they can be built, packaged, deployed and upgraded independently.

Loosely coupled means:

- Unidirectional communication of composites with backends via web services. In exceptional cases where web services are not available other well-defined APIs may be used, e.g. BAPIs. In these cases we recommend that you use web service wrappers in order to remain independent of backend systems.
- Every call creates its own transaction in the backend and there is no locking between invocations.

Composites need to abstract from the services that they are based on in order to be able to run in heterogeneous landscapes while being loosely coupled. This means that composites must be able to handle different configurations of the underlying services. The underlying services are independent of the composites therefore the composites must be adaptable to different configuration options. A minimum version of these services may be mandatory but the composites have to be able to work with any version newer than the minimum version (i.e. upward compatibility is required).

Shared volatile states and transactional locks between composites and their underlying services are not compatible with loosely coupled systems, as they would impose stateless communication.

2.2 Model-Driven Development

The model driven approach and the reuse of as much existing functionality as possible, result in less manual coding and in an increase in configured or modeled parts leading to an increase in overall productivity and quality.

Model-driven Development should be used on all layers of a composite by taking advantage of using:

- SAP Composite Application Framework (SAP CAF) for the business logic and service composition
- Guided Procedures for the process logic
- SAP Visual Composer for straightforward online user interfaces
- Web Dynpro for more advanced online user interfaces
- Composite Forms by Adobe for form based online or offline user interfaces

2.3 Enterprise SOA Compliant Architecture

Composites should be built enterprise SOA compliant. The advantage of this is that service providers and consumers are decoupled. They share the service definition, but require no knowledge of the service implementation. As a result, services can be implemented and provided without making assumptions about consumers. Additionally service consumers can invoke services without making assumptions about the service implementation.

It is recommended that composites:

- Integrate with the backend via Enterprise Services.

  The goal is backend independency to be able to integrate the composites easily with different systems as long as they have implemented the required services.

  With SAP ERP 2005, SAP delivered an Enterprise Services enabled Business Process Platform. As other vendors follow the same service oriented approach, the enterprise SOA compliance ensures high interoperability in heterogeneous landscapes.

  In addition composites reuse existing functionality and add their own business logic where standard processes fall short. This is done in a non-invasive way allowing independent lifecycles of the composite and the integrated systems.

- Have separated layers

  By defining different layers of a composite and the communication between those layers, we recommend that composites have decoupled Business Logic, UI and Process Logic and are loosely coupled to the backend.

2.4 User orientation

Composites are user-oriented applications supporting cross-system collaboration. They have a user interface, although they might contain automated process steps without user interaction.

One of the main propositions of composites is to streamline processes available in the backend and to make them accessible in a role-based, user-friendly and process-oriented way. To a certain extent, the processes even exist ‘virtually’ in the backend as a set of transactions to be used in a particular order. Composites can turn them into ’real’ processes.

It is recommend that composites:

- Provide instant usability to their users.
- Combine services from different backends and those developed in the composite to homogeneous processes and user interfaces.
- User interfaces should be task oriented, so only provide the information or require data entry relevant to the user’s current task.

- User interfaces should be process oriented guiding the user through the relevant process steps

2.5 Short-time-to-Value

Another value proposition of composite applications is the reuse and combination of existing assets of the customer's system landscape to implement new business processes. To achieve this reuse, the composites have to integrate into the existing landscape without requiring either a major upgrade or a modification of the existing backend systems (non-invasiveness of a composite). So the goal of the composites is to support existing and shipped releases of SAP and Non-SAP systems. The reuse of existing functionality contributes significantly to the 'Short-time-to-Value' goal, as existing functionality doesn't have to be re-implemented again, so that development can concentrate on the missing new functionality.

'Short-time-to-Value' can also be achieved by reducing the time to get a composite running. This time should be as short as possible and can be achieved by:

- Simple configuration
- Model Driven Extensibility
- Simple installation process
- Integration into existing system landscape without requiring additional hardware or instances (TCO)

2.6 Adaptability

As a consequence of the model-driven approach composites should be easy to change and extend. By this a higher flexibility will be provided optimizing the overall adaptability of composites. The need as well as the available technical possibilities differ for the different layers of a composite application:

- Backend Abstraction Layer: Adapt the composite to run against different backend systems
- Business Logic Layer: Adapt business objects and services to the customer's needs: Add additional attributes to pre-delivered business objects or enhance the business logic of an existing service
- User Interface Layer: Adapt the UI as required, e.g. remove and add new fields as necessary
- Process Layer: Add new steps to or remove steps from processes

3 Basic Architecture of a Composite

The anatomy of a composite (see Figure 1 for details) is a high-level description and is independent from the tools used for implementation of the different layers. Which technology to use per layer and for the communication between the layers is described in the following 'Technology Selection …' chapters.

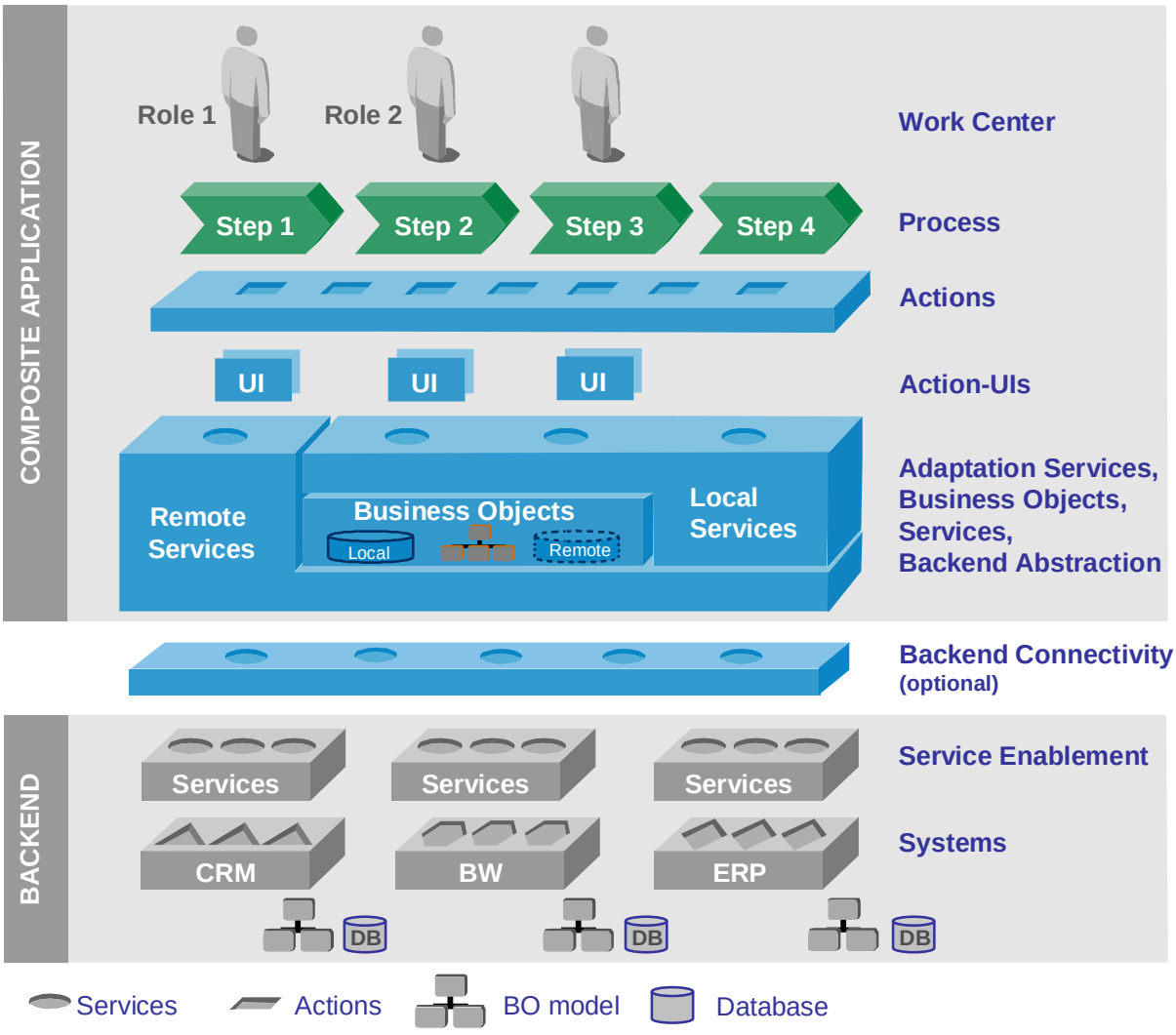

Figure 1 provides a business expert view of composites.

- **Composite Layers**
  - **Work Center**: The work center provides a role and task-oriented view of data and activities. For further information see chapter Portal Layer.
  - **Process**: A collaborative process provides composition of business services on the basis of actions and on action-UIs provided to users or user groups that are involved in the process. Within each process step exactly one action is executed (either UI or service call). For further information see chapter Process Layer.
  - **Actions**: Actions decouple process steps from services and user interfaces to allow business experts to model processes at a non technical level. For further information see chapter Process Layer.

- **Action UIs**: Action UIs provide end-user interaction with the system.
  For further information see chapter User Interface Layer.
- **Business Objects, Services**: Layer that contains business logic in addition to and as abstraction from existing business logic in the backend. Abstraction from remote business objects provides service abstraction and keeps higher layers independent from service implementation details making them replaceable.

  The business object model provides the flexibility to make transparent usage of business objects with local or remote persistency.

  For further information see chapter Business Logic Layer.

- **Backend Connectivity**

  Ideally the composite is not 'aware' of a specific backend since it is communicating via the backend connectivity layer. In the target enterprise SOA based architecture the technology of choice to connect composites with the backend is web services.

  The optional Exchange Infrastructure may be used as the messaging middleware for service communication, connectivity, transformation and portability to connect the Composite with its backends.

  For further information see chapter Backend Connectivity Layer.

- **Backend**
  - **Service Enablement:** The Service Enablement Layer provides the data and functionalities available in backend systems.
  - **ERP… Systems**: This is where the core business logic and data is available. Ideally the composite is not 'aware' of a specific backend since it uses system independent backend connectivity – see 'Backend Connectivity' above.

3.1 Service Enablement Layer

The service enablement layer provides the data and functionalities available in the backend systems, which are used in the composites by services as defined by enterprise SOA. As Figure 1 indicates, the development of the services is not part of the development of the composite itself. Instead the services have to be provided by the backend systems, which contain the core business logic and data.

3.2 Backend Connectivity Layer

As composite applications are defined as applications built on top of other applications which reuse existing functionality via service calls, the question is how the consumption of those services actually works. The recommended solution is the web service technology.

Although this might be suitable for the majority of use cases, an alternative for situations in which simple web service calls cannot meet the requirements is needed. As an example consider an IT landscape, which doesn't offer a certain business functionality required by the business logic layer to be executed by exactly one service call. Instead a series of calls, probably with parameter adaptations in-between, is necessary to achieve the goal. In this case a message broker like SAP NetWeaver Exchange Infrastructure (SAP NetWeaver XI) can be used as a messaging middleware for service communication, connectivity, transformation and portability in order to connect the composite with its backend systems. Such a message broker is useful in case simple web service calls are not sufficient and functionality such as sophisticated parameter mapping/transformation, default value handling, and message routing is needed. Typically this is not part of a composite application, it's rather the customer's responsibility to customize the application in a way that it involves communication with a messaging infrastructure. In case a message broker is not an option, hand-crafted solutions based on the service framework provided by the business logic layer are an alternative.

By default, the composite's business objects and services layer makes direct calls to the provided backend Enterprise Services and by this abstracts the calls to the higher layers (UI-/process layer). However, it is also possible for the higher layers to call backend services directly. Due to the lost flexibility this architecture

should be avoided. In the target enterprise SOA based architecture the technology of choice to connect composites with the backend systems is Web Services.

However, independent of the solution chosen for overcoming these connectivity challenges, the main idea of composites is in all cases the same: no technology related interfaces appear on higher levels (business logic, UI and process layer)! The higher levels can rely on stable interfaces which will not change in case the underlying technology, by which the services are called, changes. Following this approach a decoupling of functionality and technology will be achieved.

3.3 Business Logic Layer

Within the business logic layer the business logic and the business objects specific for the composite are implemented. The business object model provides the flexibility to make transparent usage of business objects with local or remote persistency. The service model provides service abstraction and shields higher layers from service implementation details making them replaceable. So it is recommended to make use of this abstraction in the UI- and process layer to benefit from its flexibility to adapt the final solution to different target IT-landscapes.

The business objects can have different complexity with regard to their persistency:

- No persistency in the composite.
  The business objects serve for backend abstraction and to provide services to the UI-/process layer in the right granularity. They are only persisted in the backend.
- Persistency only in the composite.
  The business objects do not exist in the backend system and are local to the composite.
- Persistency in the composite and the backend systems.
  Replication and synchronization between the composite and the backend systems is needed adding additional complexity to the implementation of the application.
  However, there are situations thinkable for which the persistency for the same business object in a distributed environment isn't an issue at all if it is ensured that the same data is only changed in one place at one point of time. This can be ensured by a proper data model or process design.

On this layer model driven development shall be used to model the business objects (attributes, nodes, and services) and generate and manage the local as well as the remote persistency of the business objects.

3.4 User Interface Layer

The user interface layer comprises online as well as offline UIs. User interfaces should in general be created on top of services provided by the business logic layer. By only using services of the business logic layer a clear decoupling between the UI and the business logic is implied.

Experiences made during the development of first composite applications showed that sometimes an additional UI-related logic layer is necessary. This layer is in-between the UI- and the business logic layer and links the UI-agnostic services of the business logic layer with the UI. It provides specialized functionality optimized for a particular UI which, without such a layer, had to be implemented in the UI itself making it hard to distinguish between mainly UI related logic (e.g. screen flow, layout, user interactions, events) and data adaptation and modification logic. Therefore the UI and this UI-adapter layer are inseparable connected.

We can approach this problem also from a different perspective: business logic services should be independent from UI's (otherwise their reuse is almost impossible). On the other hand UI's might have requirements that cannot be met by services of the business logic layer without being UI dependent. Possible mediators between the UI and business logic layer could be adaptation services provided by the adapter layer. As this functionality will be provided as services, they will typically be implemented in the business logic layer. This fact is indicated by the adaptation services being part of the business logic layer in Figure 1.

On this layer model driven development shall be used to model the User Interface screens including the screen flow and the possible user interaction.

3.5 Process Layer

Within the process layer it is defined, which process steps are executed in which sequence by which roles and how the context data of the process is passed between the process steps.

Within each process step exactly one action is executed. Actions decouple process steps from services and user interfaces to allow business experts to model processes at a non technical level. They are configurable and reusable. An action is a wrapper around a so-called callable object which either represents a user interface or a service. Such an action has importing and exporting parameters. These parameters are assigned to the corresponding application parameters. To indicate how data can be passed from a previous action (a) to the next action (b), the appropriate parameters can be assigned to each other on the process layer (output parameters of action (a) are mapped to the input parameters of action (b)).

On this layer model driven development shall be used to model the process, the process steps and the actions thus avoiding hard-coded process flows within the business logic layer.

3.6 Portal Layer

The SAP NetWeaver Portal provides a single point of access to enterprise applications, information repositories, databases and services, all integrated into a single user experience. Within the portal layer the user interfaces and processes are provided in a role-based manner using the work- and control-center concepts. The role-based content and personalization features enable users – from employees and customers to partners and suppliers – to focus exclusively on data relevant to their daily decision-making.

When a user first logs on to the SAP NetWeaver Portal, he starts in the control center. From the control center, he has access to alerts, notifications, regular work items, news, reports and other information. The user can personalize this combination of content, arranging it according to personal preferences and working needs. The control center provides an overview of all work centers a user is assigned to.
A work center focuses on a specific business area. Each role is mapped to a work center. There are more general roles of which *employee* is an example; and more specific roles of which *account manager* could be one. Within each work center is a set of role-specific portal applications rendered as iViews. A composite can be such a portal application.

4 Copyright