Composite Development Architecture Guidelines

Applies to:
SAP NetWeaver CE 7.1 EHP 1, SAP NetWeaver CE 7.2, and higher
For more information, visit the Composition homepage.

Summary
This document provides architectural recommendations for structuring and implementing composite applications. It does not focus on specific technologies like Business Process Management (BPM), Web Dynpro, or the Composite Application Framework (CAF), for example. Instead, it is intended to answer questions about how to structure a composite overall and what to consider when designing composite applications. Dedicated guidelines for the specific technologies exist and can be found on SDN.

This document includes performance and memory size measurements. These measurements are used for illustrative purposes only with no implications beyond that.

Authors: Andrey Hoursanov, Bernhard Drabant, Bogdan Vatkov, Emil Simeonov, Erik Dick, Jens Ittel, Kesavaprakash Vasudevan, Matthias Steiner, Nikolay Kabadzhov, Penka Tatarova, Ulf Fildebrandt, Ulrich Keil, Volker Stiehl, Prasanna Kumar Reddy, Harald Schubert

Company: SAP AG
Created on: 2 April 2010
Last modified on: 30 May 2011

Author Bio
Ulf Fildebrandt is working for SAP since 1998 in various areas as development architect. Over the last years he is responsible as product architect for the SAP Netweaver Composition Environment, a Java EE based product to enable customers to build application following the SOA principles.

Harald Schubert joined SAP in 2005. He was one of the early contributors to SAP NW BPM and now works as a central architect for the SAP NetWeaver Composition Environment with a focus on BPM and BAM.
## Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-04-02</td>
<td>1.0</td>
<td>First version.</td>
</tr>
<tr>
<td>2010-11-11</td>
<td>2.0</td>
<td>Second version. Contains BPM additions, new guidelines for BRM, and layout changes.</td>
</tr>
</tbody>
</table>
Table of Contents

Value Proposition of SAP NetWeaver Composition Environment .......................................................... 4

Platform Overview ................................................................................................................................. 4
  Overview of Layers ............................................................................................................................... 5
  Server Architecture ............................................................................................................................... 6
  Design Time Architecture .................................................................................................................. 6
  Programming Model ............................................................................................................................ 7

Guidelines ................................................................................................................................................. 9
  Structure of Composites ....................................................................................................................... 9
    ... In Terms of Business Data ................................................................................................................. 9
    ... In Terms of the Leading Artifact ..................................................................................................... 10
    ... In Terms of Business Processes ...................................................................................................... 11
    ... In Terms of SCs and DCs ................................................................................................................ 22
    ... In Terms of Used Services ................................................................................................................. 25
    ... In Terms of CAF Services ................................................................................................................ 29
    ... In Terms of Task Handling in a Process .......................................................................................... 30
    ... In Terms of Extensibility .................................................................................................................. 31
    ... In Terms of External Libraries ......................................................................................................... 32
    ... In Terms of Web Dynpro ................................................................................................................ 33
    ... In Terms of Business Rules .............................................................................................................. 34

Choice of Composition Technologies ...................................................................................................... 41
  ... Regarding UI Flow (BPM and Web Dynpro) ..................................................................................... 41
  ... Regarding UI Modeling (Web Dynpro and VC) ................................................................................. 42
  ... Regarding Data Transformation (CAF, BPM, SA) ........................................................................... 44
  ... Regarding Data Persistency (CAF, JPA) .......................................................................................... 48
  ... Regarding Failure Handling .............................................................................................................. 51
  ... Regarding Logging ........................................................................................................................... 51

SOA Patterns .......................................................................................................................................... 53
  ID Cross Referencing ............................................................................................................................. 53
  Asynchronous Write ............................................................................................................................... 55
  Delayed Write ...................................................................................................................................... 57
  Error Resolution: Compensation .......................................................................................................... 57
  Error Resolution: Retry .......................................................................................................................... 58
  Generic Data Transfer Object .............................................................................................................. 61

Related Content ................................................................................................................................... 63

Copyright ............................................................................................................................................... 64
Value Proposition of SAP NetWeaver Composition Environment

SAP NetWeaver Composition Environment (CE) targets two distinct areas. First and foremost, SAP NetWeaver CE enables model-driven development of own practices, also referred to as composite applications. The composites applications can be built on top of the SAP Business Suite’s best practices with the application core processes or on arbitrary backend systems. Composite applications follow the SOA paradigm of non-intrusiveness, meaning that these applications are bound to provide modification-free process extensions to the core business applications. Secondly, customers are enabled to design, deploy and run Java applications with the SAP NetWeaver Composition Environment following the JEE standards.

These goals are achieved by

- providing a highly productive development and runtime environment,
- offering model-driven technologies for rapidly translating business needs into solutions, and
- delivering an enterprise-grade application server with superior performance (see [10] and [11] for official benchmarks)

Platform Overview

SAP NetWeaver Composition Environment is designed and implemented as a usage type of the SAP NetWeaver Java stack that integrates with different components of the full SAP NetWeaver stack on various levels. This means that SAP NetWeaver Composition Environment, once it is installed in the customer landscape, leverages already existing components:

- SAP Enterprise Portal: composite applications can be federated into a customer portal via Federated Portal Network (FPN).
- Knowledge Management (KM): SAP NetWeaver CE frameworks can connect to a remote KM and its content.
- SAP NetWeaver BI: data can be retrieved from remote servers and used in composite applications.
- SAP NetWeaver Development Infrastructure (NWDI): SAP NetWeaver CE could host its own NWDI, but it is also possible to configure SAP NetWeaver CE to use a remote NWDI.

Besides a lean runtime, SAP NetWeaver CE offers a standards-based design time, the Eclipse-based SAP NetWeaver Developer Studio (NWDS). Goal of SAP NetWeaver CE’s design time is to reduce the total cost of understanding and expedite time to value by

- embracing community standards and best practices and
- providing good tool support for leveraging SAP applications through Web services
Overview of Layers

The contents of a composite application can be categorized into layers. Some parts are UI-related, some define the process flow, and others are specific to the business logic. The common functionality that all frameworks provide is the consumption of Enterprise Services:

- **Java EE frameworks (EJB, JSP/JSF):** this is the basic framework for Enterprise Java application. SAP NetWeaver CE supports all applications that are Java EE 5 compliant.

- **Composite Application Framework (CAF):** on top of the EJB framework, SAP provides the functionality to define Business Objects and services in a model-driven way. The logic is modelled, and EJBs are generated.
• Web Dynpro Foundation: most of the UI applications of SAP run with the Web Dynpro runtime. The Web Dynpro runtime is a framework that runs inside the Java EE Web container. It provides its own programming model including components, controllers, views, etc. It follows the MVC pattern.

• Visual Composer: model-driven UI-applications can be developed efficiently with Visual Composer in a completely model-driven way. The design time is available as a browser-based application running on the server or as a tool in the SAP NetWeaver Developer Studio (NWDS).

• BPM: model-driven process definitions and execution is supported via SAP NetWeaver BPM, a component that uses BPMN (Business Process Modeling Notation) as standard notation from model directly to execution.

• BRM: model-driven rules definition and execution is supported via SAP NetWeaver BRM.

**Server Architecture**

All applications run on the Java EE 5 stack and utilize its standards. This means that the runtimes on top of the Java EE 5 standard use the concepts of the Java server. The most important frameworks running on top of the Java EE runtime are:

• The Web Dynpro runtime which is integrated with the Web Container. Every Web Dynpro application runs in the Web Dynpro servlet.

• The BPM runtime running on the Java EE infrastructure. Especially for process execution, the cluster capabilities of the server are used to scale the execution of a large number of process instances.

**Design Time Architecture**

Composite applications are mainly developed in the Eclipse environment. However, there are exceptions to this rule as not all components on the server are relevant just for runtime. Complete design time solutions targeting mostly less technical users are available on the server as well:

• Visual Composer

• Rules Manager

Lately, there have been efforts to integrate Visual Composer into Eclipse to provide developers needing to work with it with a harmonized and fully integrated design time experience. *VC in Eclipse* runs locally in Eclipse and is no longer required to connect to a server to allow modelling a UI or portal content.

All toolsets within NetWeaver Developer Studio are integrated with Composite Designer. This tool provides a consistent overview of a composite application, showing the dependencies between the components of the application and checking if the contracts between them are violated.
Programming Model

SAP NetWeaver CE provides a programming model like all platforms do. There are specific frameworks for each layer (user interface, process, business logic, etc.), each featuring different entities/concepts. The picture below depicts the connections between them:
The frameworks contribute the following entities to the overall programming model:

- **BPM**: provides the process and task definition. A process can use services and tasks. The task itself can use user interfaces.
- **Web Dynpro**: provides a Web Dynpro application that can run standalone and the Web Dynpro component as reusable entity. Web Dynpro components can use services.
- **Visual Composer**: provides the application and the views in Visual Composer models. Visual Composer models can use services.
- **CAF**: provides the CAF application service as an adaptation component to adapt data and the CAF Business Object to locally store data. The CAF application services are exposed as Web services and can compose other services.
- **Service Composition**: provides Service Composition models that are exposed as services and can compose other services.
- **Java EE**: provides EJB technology to implement business logic and JPA to persist data locally. Every EJB can be exposed as a service.

The service notion is very important in the Composition Environment. In the local case for example, CAF application services and Service Composition, are exposed as Web service or EJB. In the remote case, SAP NetWeaver CE can consume services that are available as Web service or RFC.
Guidelines

Structure of Composites

A composite application usually contains a multitude of components (see the definition of the programming model above), assembled in development components and software components. The frameworks within CE allow for building powerful applications, but attention must be paid while structuring a composite application for ensuring that developer productivity, performance, and maintainability are not adversely affected. The following guidelines therefore explain basic principles for structuring a composite application.

... In Terms of Business Data

Addressed issue: a composite application contains a lot of artifacts from different domains. Many of these artifacts can be used to store information. The appropriate type of storage will be described here.

The programming model of SAP NetWeaver CE provides various domain frameworks for specific purposes:

- Definition and execution of business processes: SAP NetWeaver BPM
- Definition and execution of user interfaces: Portal, Web Dynpro Java, Java EE (JSP, JSF) and Visual Composer
- Definition and usage of Business Entities: CAF, Service Composer, and Java EE (EJB, JPA)

A Business Entity is not (yet) a concrete modeling artifact in the CE landscape, but a conceptual abstraction for a certain business concern (from a consumption perspective). In SAP NetWeaver CE, we perceive the Business Entities mainly as data centric artifacts and as mediators providing services that operate on the associated business data.

In case that you opted for the loosely coupled approach, utilize the Business Entity concept to transform business data from the type system of the service providers (e.g. SAP Backend BO) to your canonical data type system in the composite context and vice versa.

Depending on the chosen implementation technique, Business Entities have different characteristics. Very prominent implementation/usage patterns are:

- Provide an intermediate data storage for the local execution context. (CAF, Java EE)
- Provide facades to access a single or multiple more complex entities in a simplified manner, such as Enterprise Services (Business Objects). (Service Composer)
So, there is a design decision at the beginning of the composite application development on where to store the data. Both options are possible and it depends on the business logic to determine which one is appropriate. The local persistency of data is the preferred option if it is acceptable to have a copy of the data in the composite application. Keeping the data in the composite would mean that the composite application has to synchronize the data with the remote systems at a later point in time. If the data in the backend systems always has to be consistent and a local copy is not acceptable, then the data can only be persisted in the remote systems.

SAP NetWeaver BPM and the UI technologies also provide their own method for managing data in their context, which can be used to store/transport data from the business context. There are therefore various options for dealing with the data and the characteristics of the storage concept:

### Storing data in the BPM process context:

A data object in a process holds data that has to be potentially available for all activities in the process. Data is passed around through input mappings from data objects to activities and output mappings from activities to data objects, respectively.

**Scope:** Data in a process context is persistent and available for the lifetime of a process instance.

### Storing data in the UI component context:

If the composite is a UI application without a process surrounding it, it is a valid option to store the data in the context of a Web Dynpro application. This option is only possible if the data does not need to be persisted on the server side, i.e. does not need to survive a server shut down or crash.

**Scope:** Data in a Web Dynpro context is transient, available during the lifetime of the UI component, i.e. a user session.

### Storing data in a business entity, here CAF:

Business Objects in CAF can be used as persistent storage of data that has to be available across multiple processes and UI applications. Here, the life cycle of the content is not bound to a UI session or process instance. It has a life cycle of its own.

**Scope:** Data in a CAF Business Object has its own lifetime, is persistable and is not coupled to anything else.

Driven by the concept of separation of concerns, but also based on performance considerations, we recommend that the amount of business related information stored in the process context is kept small. Try isolating the business data and maintaining it in a business entity instead.

**In Terms of the Leading Artifact**

**Addressed issue:** A composite application contains a lot of artifacts from various domains. Many of these artifacts can be used to store information. This section explains how to provide consistent access to the aggregated information.
Identifying the Leading Artifact

During the definition process of the composite, it is important to think about the leading artifacts of the composites. A leading artifact is an entity in a composite application that is accessible during the whole lifecycle of a composite and can hold the data, so that other entities can access the data and do not have to replicate all content.

This means that the composite application designer has to decide which would be the leading artifact of the particular composite or a part of it. This is essentially a question of the scope of the data and the lifecycle.

- If the process definition is the most important entity, the natural choice is for the data to follow. As a consequence there would be no local persistency (after execution of the process instance).
- If the data is important and should have its own lifecycle, a CAF Business Object is a good choice.
- If the data is only relevant in one UI task, the data can be stored in the Web Dynpro context.

As usual in application development, there could be a mixture of these approaches, and this is not a contradiction. The only decision to take is how important a specific part of the application is, for example if the data simply follows the process definition, if the data should only be available in a user interface, or if the data should be persisted in the database.

Nevertheless, there are some possible overlaps. It would be possible for example for the data to be persisted in a CAF Business Object or the process context. Both solutions therefore have advantages and disadvantages:

- Arguments to store data in the process context
  - Data that is required in the whole process is connected to the process itself by defining it in the process.
- Arguments to store data in the CAF Business Object and reference it in the process via ID
  - The data can be accessed independently of the process instance. Data has its own lifecycle.
  - Large amount of data is not transferred between activities. Only the ID is passed between process steps.

... In Terms of Business Processes

The following considerations and guidelines are mainly focused on SAP NW BPM.

Controlling Model Complexity

Control Flow

SAP NetWeaver BPM offers a graphical design tool. The process graph shows control flow and data objects as well as their usage in process steps. This allows for a quick overview of the process flow in general if the size of the process remains manageable. Manageable means that the process designer does not have to simultaneously understand the interaction between more than 25 – 50 graphical objects at a time. Our first recommendation is therefore:


If your model grows beyond that, there will be clear candidates for reuse, i.e. portions of your process that encapsulate a meaningful (set of) feature(s) worth putting into a separate process. Use sub-processes in SAP NetWeaver CE 7.11 in the form of ‘referenced sub-processes’ wherever possible to encapsulate reusable functionality. You can also leverage ‘embedded sub-processes’ in SAP NetWeaver CE 7.20 to manage the complexity of your process model even if your sub-process is not intended for reuse.

Be aware that there is also a runtime aspect to this hint. The process visualization (graphical log) will look as complicated as your design time model, but in addition has instance data attributed to it. The business log (a textual representation of your process instances) needs to convey this information in textual form - there it will be even harder to see complex relationships between steps.
Closely related to the model complexity in terms of size (number of steps) is the shape of the model in terms of its graph. A nice, compact graph attracts a second look whereas a large, chaotic graph will easily distract from what it wants to convey: the idea of your business process in a single picture. BPMN is a very powerful graphical notation that you should use with care. One feature in particular tends to let people lose the overview of a diagram if used too excessively, hence our next recommendation:

**Model Block-Oriented Wherever Possible.**

Taking advantage of the BPMN means that you utilize the *flow-oriented* modeling technique, which is well suited to describe real business processes. Nevertheless the *block-oriented* modeling concepts can come in handy for some reasons: A block is a piece of diagram with one entry line and one exiting line. This makes it a portion of a diagram that easily lends itself for refactoring, that can for example be easily converted into a sub-process and be easily copied from one part of the diagram to another. Block-oriented diagrams with their typical nesting and lack of overcrossing lines tend to make diagrams look simpler and convey the existing structure more easily. BPMN allows you to connect any step in your diagram to any other step: this feature tends to clutter the diagram and, if source and target are not on the same screen, easily lets you lose track of the control flow. It should therefore be used with care.

**Data Flow**

The BPMN diagram clearly shows the control flow for your process. The *data flow* is only visualized for top-level data elements however. In order to find out where a certain attribute of a large structured data object is populated, where it is manipulated and which steps it is actually consumed in requires navigation to each individual step and inspection of the input and output mappings.

Hence, it is advisable to carefully plan the data context of the process (*global data*) and the interface/data context of each step. A process designer must consider which data is actually required to *drive* the process (e.g. status variables that are needed to make decisions in the process, deadlines that need to be monitored, relevant user information). We call this *primary data*. Opposed to this there is *secondary data* which is only *carried* by the process because the process acts as a data mediator between steps (e.g. details read via a service call that is then passed to a UI) or as a convenient generic persistence layer.

**Name Your Primary Data as Such**

Use meaningful names, comments or your own conventions to achieve this. Primary data, in particular status variables must be understood by anybody who attempts to use, manipulate or refactor the process. Overlooking or misinterpreting primary data will result in difficult to catch application errors and misuse of the process.

**Minimize the Amount of Secondary Data in Your Process**

If you reuse services or UIs, you will have to work with the given set of interfaces that you will have to populate or interpret, meaning that your choice of data you have to provide or accept is limited. You should however be able to use reduced (*thinned out*) data structures within your process that only store the attributes that you actually need. In SAP NetWeaver CE 7.20, you can use service adaptations offered through Service Composer to *thin out* vast interfaces to the data you really require. See the performance and data volume sections to fully appreciate the meaning of this recommendation. See the concept/notion of Business Entities, too.

If you are in a situation where you determine the interfaces of services and UIs yourself, for example because you design them as part of your project, you should do everything you can to adhere to this guideline. In particular:

**Design New Interfaces with Small Signatures**

Let these interfaces expose less than 25 parameters and prefer copy-by-reference to copy-by-value. While the first part of the recommendation is self-explanatory (smaller interfaces = fewer dependencies, less complexity, less data overhead), the second part merits an example as it is valid both for services and UIs. Passing references (e.g. to Business Objects) typically works well if the receiving application can interpret the reference and can access the referenced object’s data effectively. Passing references typically means that you transport only the ID/key of a Business Entity between different participants in the flow (see also [...]
In Terms of Business Data] and [Minimize the Amount of Secondary Data in Your Process]). For example, if you pass the reference to a service contract, the contract number for example, to a UI, and the UI application can use this key to read the required details of the contract, like the issue date, the contract value, etc. without a performance penalty. As a consequence, the UIs interface contains only the contract number instead of the 75 attributes that a fully fledged contract Business Object might well have.

Copying data (as opposed to referencing it) has several other unwanted implications, such as:

- The copied data in your process can run out of sync with the original object’s data. This can cause the process to make incorrect decisions, as it is based on outdated information.
- The copied data can pose a security risk if it is sensitive. In releases 7.11-7.20, SAP NetWeaver BPM does not offer fine granular access rights to process data. If you are allowed to see the process instance, you are allowed to see all its data. Not every contract value or every business decision is suitable for viewing by an administrator. A service or a UI always implements dedicated security policies that are more fine granular (as they are specific to the given Object at hand) than a generic business process infrastructure.
- The amount of data that is generically stored with your process grows with the size of your data context. See the performance section for further impact.

Keep the Number of Attributes to be Mapped at any Interface Below 50

Finally, a large data context or a large activity interface requires large data mapping definitions. BPM supports a graphical mapping tool that can visualize even complex mappings efficiently. In practice however, there are limits to the efficiency of any graphical tool.

If you find that more than 50 attributes are to be mapped, apply the [Design New Interfaces with Small Signatures] guidelines. A mapping designer will easily lose track if a mapping does not fit onto one screen. Refactoring and maintenance of mappings like this are problematic: did I really need attribute X? How does the underlying UI interpret attribute Y? What is the correct mapping function for attribute Z? In some BPM projects, the mapping consumes up to 50% of the time needed to design a process. This is a clear indicator that simplifying the data structure will reduce project costs in at least the mid to long term.

Another aspect must also be mentioned here. SAP NetWeaver CE allows you to design a composite application very easily using BPM. One reason for this is that the BPM tool provides a generic persistence – at no extra design cost, all attributes of a business process will be saved automatically whenever the corresponding process instance reaches an automatic save point. Rollbacks, transactions handling, etc. are taken care of by the process engine. As with any automatism, there are limits. This guideline is strongly related to the [] guideline.

Performance Considerations

... at Design Time

In many ways, design time performance is directly related to the size of your models. If you use all features of CE at once, meaning the process editor, UI designer(s), Composite Designer, etc., a large number of elaborate components will be competing for CPU and memory resources. Significant improvements in terms of to memory consumption in NWDS have been implemented in 7.20. Focusing on BPM, it is worth mentioning a few simple hints that can help you to significantly reduce the memory footprint of Process Composer.

Use ‘Move-Corresponding’ Wherever Possible.

This is especially true for large structures. A mapping definition is represented as a model at design time. The complexity of this mapping model can be quite significant when a large number of individual attributes is mapped from one BPMN artifact to another; e.g. from data objects to activities or from events to data objects. In SAP NetWeaver BPM, move-corresponding is a convenient way to move data between identical data structures. In 7.20, move corresponding has been condensed to one single mapping command, regardless of the size of the structure to be mapped. In addition, if you design a canonical type system for your composite and you either deal with high load or if you have a large number of mapping definitions, make this feature part of your considerations.
In 7.11, a move-corresponding was expanded at design time, meaning that a set of mapping statements is recursively generated to map the entire structure. The difference in memory footprint of mapping a larger structure (> 50 attributes) in the input and output mapping of a human activity when using move-corresponding can go down by a factor of 10 when using the 7.20 design time. Memory consumption improves accordingly.

**Reuse Human Tasks Wherever Appropriate.**

A *task* in BPM is a re-usable object. As BPM implements the full WS-human task compliant status model, a task should be imagined as a rather complex object (which it is). Reusing tasks as opposed to copying them has a significant (positive) impact on memory consumption in a BPM project.

**Housekeeping at the IDE.**

It is advisable to constantly clean up your projects in the IDE to keep the performance of the NWDS on a good level. Close or remove artifacts in the IDE that are no longer referenced by active processes in your runtime-systems. Such artifacts would be:

- WD/VC UIs
- Service endpoint definitions
- Individual task definitions
- Rules

Please note that removing artifacts from a processes model (or referenced DC) is an operation that should be executed carefully. If you deploy such reworked artifacts to the server which has still active instances based on the preceding definitions it is likely that the deployment will invalidate these instances. This is considered to be an incompatible change. Especially be careful with service endpoints (i.e. event triggers).

For UI components a reimport feature is provided that allows to re-read its definition for a task UI. Again ensure that you do not perform incompatible changes (e.g. remove parameter at I/O of the UI), otherwise running instances on a server might break at the point when you deploy the new definition. If only compatible changes are done to the UI component, the system will preserve the existing mapping at the I/O interfaces from task to task UI.

There is currently no support from the system to detect incompatible changes automatically.

**... at Runtime**

Runtime performance here means factors that influence the CPU and main memory consumption of the process engine that runs on your Java server and executes the instances of your process model. There are quite a few factors like this that you can influence with the design of your process model. A few theoretical remarks about the BPM process engine are required to elucidate the following guidelines.

The BPM engine uses an in-memory algorithm to efficiently share resources (memory and CPU) between all process instances currently in execution. On a clustered installation, one engine instance will run per node. Elaborate load balancing, communication and fail-over mechanisms are in place to ensure efficient use of the full cluster resource.

The engine executes process instances in a transactional and fail-safe manner. This means that the state of a process is stored in the data base at *save points* whenever the process logic, technical constraints, or general monitoring requirements demand it. As the engine executes an arbitrary number of different process models, all with different data context definitions, it cannot use dedicated (transparent) DB tables to store this data. Instead, at every save point, the data context of a process is serialized to XML and stored as one ‘blob’. When the data needs to be read back, it is fetched from the DB and parsed to reinstantiate the data objects in the memory. This engine-persistent storage is required to enable fail-over of process instances in case of a Java engine or a hardware failure. If this occurs, the engine will simply reload its previous state and continue process execution from the previous save point.

Whenever a context switch occurs for a process instance, the entire instance must be serialized on the node where it is currently executed and later deserialized on the node where execution is to continue. A context switch like this can occur when an incoming request is issued on a node in the cluster where the process
instance is not currently running. Requests of this kind are: service requests, a BPMN event (via correlation) or, a human interaction leading to a status change of a task.

For auditing reasons, note that there are at least two save points for every activity in your process: one after creation when the input data is available and another one after completion, when the output data is available. Human tasks can have significantly more save points than automated activities, typically one after each status change (e.g. task created, task started, task claimed, task failed, task completed ...). Also note that all major changes to a process instance and its activities (including changed process data) are written to the business log, where they are the basis for providing information about the history and the current state of the process (in graphical or textual form).

In memory, data objects are represented using a standard compliant SDO implementation.

The size and complexity of the data context of a process therefore directly impact engine performance in three ways:

- The data context of every process instance currently in execution must be kept in the main memory. Very large data contexts (> 1 MB) can significantly impact the number of process instances that can be simultaneously executed.
- When a save point is reached, the process instance must be serialized and written to the database. Serialization consumes application server CPU time, and the serialized data stream consumes DB server CPU time and DB space.
- When a process instance is re-loaded from the database, its data is de-serialized, and the Java runtime objects are re-constructed.

To underline the importance of the [] guideline, consider that each attribute in the global data context is serialized and stored in the business log whenever it is changed. Each activity in the process has at least two save points, where the full input and output data (respectively) are serialized and stored.

**Carefully Design the Data Flow for Parallel Execution**

A data object can easily become a synchronization point, thus hamper the system performance, if it is modified from different artifacts of the process model concurrently. That is especially to be considered when using parallel split gateways or dynamic parallel loops (par-for-each) and trying to store the outcome of the parallel operation into a single/central data object. For example, assume a simple process to complete attributes of a PO:
Both automated activities would access the data object (DO) ‘PurchaseOrder’ for some values of the input mapping and would also like to store the outcome of their computation into the ‘PurchaseOrder’ via an output mapping. In such a case, the execution of the activities would not take place in parallel, but rather sequential (with no guaranteed order). This is due to the fact that both activities would request a write lock on the DO ‘PurchaseOrder’ for the complete execution, but only one of them could get it at a time. The other has to wait until the first one completes its operation before it gets the lock and can start its execution. So the overall performance gets affected. Instead of having \( t = \max(\text{T(DetermineReceivingGate)}, \text{T(Calculate Tax)}) \) it would be \( t = \text{T(DetermineReceivingGate)} + \text{T(Calculate Tax)} \). Please note that the lock is always on a complete DO and not on individual artifacts. An alternative would look like this:
In such a model there is no blocking. Both activities require only a read lock on DO 'PurchaseOrder' for input mapping, which can be granted concurrently. Since they are mapping their outcome to different DOs the requested write locks to not lead to a blocking situation for the activities. Of course the computation result has to be stored in the DO 'PurchaseOrder' again. This is achieved with an additional mapping activity directly after the join of the parallel execution. So \( t = T(\text{DetermineReceiving Gate}) + T(\text{Calculate Tax}) + T(\text{Mapping 0}) \).

The same issue can appear if you mark a single activity for dynamic parallel execution (par-for-each) and perform an output mapping to the same data object. Again, the execution would be serialized due to concurrent request for a write lock.

Here, the proposal is to model an embedded sub-flow and execute this flow dynamically in parallel, as in this example:
Additionally, in the sub-flow you have to separate the concrete activity from the mapping to the central DO. Again, use an intermediate DO to store the computation result, i.e. to avoid a serialization by a write-lock on the central DO. Then map the intermediate DO content to the central DO. Of course, the system will serialize the execution at the mapping activity. But since mapping is typically a much less expensive operation than the 'real' activity, the lock won't last long. This means the negative impact of the serialization at the DO is essentially less than in the first approach.

**Carefully Consider the Use of Massive Par-For-Each**

The par-for-each looping construct allows iterating on a collection of items in parallel. Consider the following diagram showing a process that processes a large number of order items by invoking a service for each of them:
As opposed to the previous example, there is no serialization happening on the data object as all we do is read it and read locks are non-exclusive. To understand why this scenario needs to be handled with care, again, some more understanding of the BPM process engine is required.

The process engine allocates one *token* for each parallel branch within a process instance. For instance, if you start a process instance, there will be initially exactly one token; when you branch out using a parallel split gateway, an additional token is created; once you synchronize them back using a parallel join gateway, one of the tokens gets consumed again, and so forth. The par-for-each construct works in the exact same way: if you iterate over n items, n-1 additional tokens are created for the duration of the looping.

A token doesn’t mean thread in the operating system sense. The process engine has a configurable amount of threads managed in a pool that it uses to execute process instance tokens. Threads are assigned to tokens in a fair manner. Now, while the number of operating system threads remains fixed as the number of tokens increases, each token incurs a certain overhead in the process engine, comparable to the allocation of memory needed by the operating system to allocate a new thread ([17]).

Also, do note the par-for-each construct does a defensive copy of the items to avoid concurrent modification issues (items could be added or removed in parallel by another branch in the process). This leads to additional memory overhead. And to complicate things even more, do remember that the activity being looped could be not only an automated activity like in the given example but even something as expensive as a sub-process or a human activity/task.

All this effectively means an excessively large number of items in a par-for-each loop can lead to out-of-memory situations. For this reason, it is recommended to limit the degree of parallelism. The exact number in the end depends on the sizing and resource utilization within the system but a best practice is to keep the number of items processed in parallel at around 10.

Following up on the above example, the order items could be chunked in groups of 10 as follows:

---

1 This notion is technically somewhat imprecise but sufficient for this discussion.
Prefer Callbacks Over Pull Patterns

Avoid busy waiting patterns wherever possible. Consider the following example where a process waits for a backend system to be ready with some task before continuing. It repeatedly invokes a service provided by the backend to see if it is ready. If it is not, it waits a certain amount of time before polling again.

![Process Flow Diagram](image)

This approach has a couple of implications. For one, it puts unnecessary load on the engine that can be avoided by using a callback. Each time the process does some processing, even if it is just a service invocation, the process engine has to write save points to the database, log meaningful entries to the BPM Business Log for auditing reasons, etc. If a large number of such process instances run on a server, this can significantly degrade the overall system performance.

So, where possible, employ a callback from the backend (with a proper correlation condition) as shown below.

![Callback Flow Diagram](image)

Note that irrespectively of the approach taken you are highly advised to think about the edge cases where a backend is unavailable, for example. While in the first case the process would keep polling forever, the second version with the intermediate message event wouldn’t come to an end either. Consider using strategies as in guideline [Employ Deadlines to Increase Process Transparency] to compensate such situations.

Use Reporting Activities for Reporting Purposes and not as a Business Data Store

Reporting activities are used for persisting data that can later be leveraged for analytical purposes such as reporting. Data is persisted in the BPM Business Log, which is essentially a generic DB table shared across all process instances. Due to its nature, the BPM Business Log is not recommended for storing arbitrary business data. Use CAF or JPA/EJBs instead in such cases.

Narrow Down Potential Owners as Much as Possible

Each task has a list of potential owners identifying those users that are allowed to claim the task. Each potential owner identified in this way incurs a certain amount of runtime overhead (task list creation, task management, DB load by logging to BPM Business Log). Hence, it is highly advised to restrict the number of potential users where possible. Proper domain modeling is required. Avoid having generic tasks that are open to everyone.

Also, the higher the number of potential owners, the higher the risk that there are two end users trying to claim the same task (leading to reduced end user satisfaction). Another benefit is that you don’t overcrowd the users’ inboxes.
Reliability Considerations

Avoid Failure-Prone Logic in Expressions

When modeling processes, make sure not to include error-prone logic inside expressions (e.g. as part of mappings). Specifically, refrain from invoking backend services wrapped through EJB functions as the services might be unavailable at the time of invocation. Consider the following examples that both invoke a backend service:

The first version invokes the service through an EJB function that wraps the service call. The second uses a regular Web service. Left aside the fact that the second version better conveys what is actually happening, it also has several practical advantages. For one, it allows capturing errors happening during the service invocation and reacting accordingly (catching faults via a boundary event). Service invocations via automated activities also underlie a retry mechanism that will try re-invoking the service several times before putting the process into failed state. If the process then finally reaches failed state, this doesn’t halt the entire process instance but only affects the branch in which the service was invoked. Other concurrently executed branches, e.g. involved human activities, may continue executing. If a function invocation goes wrong, the entire process goes into error state. Also, errors in expressions are harder to trace.

Employ Deadlines to Increase Process Transparency

Processes often involve service calls as part of their execution. As for the most part this means crossing system boundaries, it can often happen that one or more backend systems are unavailable or irresponsible while the process engine continues to operate. Respective process instances are then silently set to failed state and end users, specifically if this happens between consecutive tasks, will assume the process has wreaked havoc and reach out to the administrator, asking why the process is not continuing:

It is a good practice to monitor such (possibly user-perceived) “critical” areas within a process and to automatically notify administrators, ultimately increasing process transparency and user satisfaction. The following is an example of how the above process can be changed to achieve the desired behavior:

Parent process:
Sub-process:

In this example, the backend service call to be monitored was essentially wrapped in a sub-process that spawns of a parallel branch right before the service is called. If the service call succeeds before the deadline is reached, the sub-process terminates and consumes all tokens, implicitly cancelling the deadline branch. Otherwise, the deadline is reached and a non-critical escalation event is triggered. Non-critical at that point means that the parent process can handle the escalation in a separate branch, sending out a notification to the administrator in this case, and avoiding premature termination of the process. So, if the administrator brings the backend back up and then resumes the process, it can still end regularly. The parallel split and mapping activity after the deadline timer event are used to re-raise the escalation. This is a common practice, often combined with some sort of exponential backoff to prolong the time intervals between successive escalations (e.g. 1st escalation after 5 mins, 2nd after 15 mins, 3rd after 1hr, etc.).

Sizing Considerations

A dedicated sizing guide for SAP NW BPM starting with release 7.20 is available on the SAP Service Marketplace. See [16] for more details.

... In Terms of SCs and DCs

Addressed issue: the artifacts of a composite application can be structured in many different ways according to the SAP Component Model. This is an explanation of how to structure a composite application.

During development of a composite application, one of the most important questions is how the application will be structured in projects, in particular how Software Components (SC) and Development Components (DC) will be used.
Software Component Granularity

As an SC defines the deployment granularity, it is usually created if parts of the composite application should be executed on a different server or if parts of a composite application can be deployed independently, for example if some of the functionality is optional.

Every SC defines a distribution archive. SC granularity is therefore driven by the degree of distribution that is to be achieved. Creating different SCs only to bundle business-relevant parts together (or separate them) is definitely not recommended. If the parts of a composite application should be executed on one server and there is absolutely no intention of deploying them on different servers, the content should be put in the same SC.

If the application design is done in such a way that different parts can be distributed independently, it is also recommended to put those parts of the application in different SCs, because this is the smallest distribution granularity in software logistics. The SC is also the entity that is versioned, so the SC is the entity that is meant to structure the deployment.

Development Component Granularity

Different CE frameworks provide different DC types. It is therefore a good choice (and sometimes the only option) to put the content for different domains in separate DCs. This means that all process content is located in a Process Composer DC, all Web Dynpro content is part of a Web Dynpro DC, etc.

A DC is the atom of reuse, i.e. the smallest undividable functional piece. The composite application developer therefore has to plan for reuse and proper segregation of reusable pieces into a minimal number of DCs. For generic functions, individual DCs should be created to facilitate their reuse and to collect all required elements of the function in this DC.

This means that parts that are meant for reuse should be put into separate DCs. You should also reserve enough time for planning the API and separating API and implementation. You can then add the API to a public part of your DC.

Choose the DC structure according to its function and not according to the organizational structure of the responsible developers. Choose the DCs in such a way that the involved developers are working in the same team and at the same location. Only if these conditions are met can you use the inactive state of objects, which is, from an organizational point, mandatory for distributed responsibilities. These two statements seem contradictory at first, but they should really be used together, so that together they define the minimal structure of DCs.

After a change, the entire DC must be rebuilt. You should therefore choose the size carefully so that it does not contain too many objects. This is a very flexible statement, therefore some more details are required.

Build Time

The following section provides more details regarding the general recommendations about how to structure content and what these recommendations look like in the light of build times. To illustrate how the structure influences the build times, Web Dynpro was chosen as an example.

Web Dynpro recommendations to structure the content are defined as follows, taken from [2]:

- Optimize the development performance with the best deployment granularity
  - Keep your Web Dynpro development components (Web Dynpro DCs) as slim as possible to accelerate build/deploy/run turnaround cycles.
- Optimize the application architecture with the best Web Dynpro component granularity
  - Rule of thumb: implement one business task in one Web Dynpro component.
  - Don’t make Web Dynpro components too large
    - Distributing work between several developers is harder
    - Reusability is very poor
  - But: don’t make Web Dynpro components too small
    - The application might not perform well
    - The resulting ”sea” of components might reduce maintainability
• Reuse level too small
• Higher system overheads

• Apply Web Dynpro component separation principles
  o Create a single root component which embeds and arranges separate UI components.
  o Separate the UI in visual Web Dynpro components.
  o Centralize model access in a faceless model component and use this singleton model component in other components by applying the component usage referencing mode.

• Apply Web Dynpro Development component separation principles
  o Place Web Dynpro models, components, component interface definitions and local dictionaries in separate Web Dynpro DCs.
  o Distribute work between developers by assigning one DC to one developer.
  o Weight the benefit of applying the principles to the overhead they might require.

The last sentence in particular indicates that there is more behind the separation than just the theoretical guidelines. A very interesting point in this area is that the build times are to some degree related with the number of DCs. The following table summarizes the situation for the same functionality, either located in one DC or in four DCs. The measurement is performed on one PC, so the numbers are comparable to the others, but the numbers are not a guarantee that every PC with the same performance achieves the same.

<table>
<thead>
<tr>
<th>Web Dynpro DC Scenario</th>
<th>Build Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A WD DC consisting of 4 WD components</td>
<td>31.609 seconds</td>
</tr>
<tr>
<td>\all</td>
<td></td>
</tr>
<tr>
<td>Four WD DCs containing one WD component each</td>
<td></td>
</tr>
<tr>
<td>\change</td>
<td>7 seconds</td>
</tr>
<tr>
<td>\create</td>
<td>6.328 seconds</td>
</tr>
<tr>
<td>\result</td>
<td>11.141 seconds</td>
</tr>
<tr>
<td>\search</td>
<td>25.578 seconds</td>
</tr>
<tr>
<td></td>
<td>50.047 seconds (total)</td>
</tr>
</tbody>
</table>

What becomes very obvious is the fact that the build of a DC has some overhead. Opting for too many DCs will therefore create problems during build time.

The structure of the DCs of a composite application is therefore always a compromise between the guidelines for separation of business logic in DCs (as small as possible according to the business needs) and the overhead of the infrastructure for each DC. There is normally a conflict between optimizing either the number of DCs for the fastest overall build (as few DCs as possible) or the fastest build for developers (as many DCs as possible). The numbers above should provide some guidance about how much the additional overhead for development would be. If the decision goes to the more coarse grained DC design, the build times for developers increase. If the decision is to use fine granular DCs, the overhead of the infrastructure (build time and maintenance for DC dependencies between the DCs) increases. It is therefore important to make a balanced decision.

The actual guidelines are highly dependent on the application use case, but the build time and infrastructure aspects still have to be taken into account during the decision process of the composite application structure.

**Clustering of Content**

Usually a composite application contains a lot of entities of one framework, like Web Dynpro components or EJBs. The recommendation is to put entities that are highly related into one DC. This does not mean that all components are put into one DC. In principle, it is a question of how to cluster the components in DCs. A cluster belongs into one DC. If the connection between two potentials clusters is very weak (only one or two dependencies), the components are put into two DCs. This recommendation can be defined as follows:
• If a set of components is using the same set of external components, the set of components should be put into one DC. They belong to a cluster of components.

![Diagram](https://example.com/diagram1.png)

• If a set of components is grouped together and only one component is accessible from external (defining a public API), the components should be put into one DC. They belong to a cluster of components.

![Diagram](https://example.com/diagram2.png)

At the start of the composite application design process, you therefore need to think about the components and how they are connected.

---

**... In Terms of Used Services**

**Addressed issue:** a composite application usually uses services from different other systems. This section describes how these services can be used.

**Loose Coupling or Tight Coupling**

Usually, when the application design is done regarding the business functionality and the user interfaces that are required, the next question is how the services that are needed can be consumed. Sometimes the available services are not sufficient to fulfill the business requirements of the composite application and therefore services have to be defined and implemented in a remote system. This design step is not described here, neither is the definition and implementation of these services. After this is clarified and the set of services to be used by the composite is identified, the obvious next question is where the data of the service calls is used. And here the first architecture decisions will be made, because SAP NetWeaver CE frameworks, like Web Dynpro, BPM and CAF, provide the capability to consume services directly. The first decision now is whether a service should be used directly or not. This section will focus on this question and provide some guidance about how this can be done and what the advantages and disadvantages are.

In general, SAP recommends implementing remote consumption of business functionality using loosely coupled, asynchronous, stateless communication using Web services, see [12], [13], [14], [15].

The term loosely coupling gives you an indication about the design principle for an application regarding service consumption.

Very often the discussion is reduced to the question of whether an application talks synchronously or asynchronously to its peers. Although this is certainly one aspect, it doesn't cover all relevant dimensions of loose coupling. The goal of loose coupling is to reduce dependencies between systems. Therefore, in order to provide a definite answer as to how tightly an application is coupled to other systems, you can be guided by a second simple question: What are the consequences for system A (the calling system) if you make changes in system B (the called system)? Most probably, this question already reveals a large number of
dependencies between your application and others that go beyond a pure classification of the communication style between them. There are a few more assumptions about application design that affect coupling between systems:

- **Location of the called system (its physical address):** Does your application use direct URLs for accessing systems or is your application decoupled via an abstraction layer that is responsible for maintaining connections between systems? The service group paradigm used in SAP NetWeaver CE is a perfect example of what such an abstraction might look like. Using an ESB is another one.

- **Number of receivers:** Does your application take care of the receivers of a service call? Or does it simply drop a message "somewhere", while other mechanisms take care of transporting the message to the receiving systems? Don't underestimate this assumption! If you take the "loosely coupled" approach, it actually means that you are not making any assumptions about the systems you are talking to. This implies a completely different architecture compared to tightly coupled applications. You never know when or if a service call returns due to the number of involved systems, and your application must be prepared for that!

- **Availability of systems:** Does your application require that all the systems that you are connecting to are up and running all the time? This is obviously a very hard requirement, especially if you want to connect to external systems that are not under your control. If your answer is "Yes, all the systems must be running all the time," you are obviously tightly coupled in this regard.

- **Data format:** Does your application reuse data formats as they are provided by the backend systems, or do you use a canonical data type system that is independent from the type systems used in the called applications? If you reuse the data types from the backends, you probably have to struggle with data type conversions in your application. This is not a very loosely coupled approach.

- **Response time:** Does your application require the called systems to respond within a certain (acceptable) timeframe, or is it acceptable for your application to receive an answer minutes, hours, or even days later?

There are even more dependencies, but the message should be clear: Loose coupling is not one-dimensional. For each of the aforementioned aspects of loose coupling, you have to make decisions. And they are not easy because moving toward loose coupling has serious implications for the architecture of your application. So loose coupling comes at a price, especially in terms of complexity, and you have to decide whether you want to pay the price for it.

The benefit of loose coupling is flexibility and agility. If you are aiming for a loosely coupled approach, you will get unparalleled flexibility for adaptations to changing landscapes. Since you aren't making any assumptions about the landscape your application is running against, you can easily adapt it as needed (provided your frameworks and tools support you like SAP NetWeaver Composition Environment does). This is especially important for partners and ISVs who can develop applications once and easily install and configure them at their customers' side. The application itself stays untouched.

It isn't just partners and ISVs who will benefit from this approach. It is useful within companies as well: Once you've established a successful new application, you will most likely want to reuse it within your company in other locations or regions. Very often, the IT landscape in the new locations differs from the one the application was originally designed for. If you take the loosely coupled approach right from the beginning, this undertaking will not frighten you. Another aspect you should consider is the probability of landscape changes during the lifetime of your new application. Due to mergers and acquisitions, or due to system consolidations, the landscape underneath your application is constantly changing. If you are not prepared for loose coupling, you'll be forced to adapt your application again and again.

### Concrete Implementation of Loose Coupling in Service Consumption

The concrete implementation of the loose coupling principle in a composite application can be performed as follows:

- Enterprise services are used only via WS proxies in EJBs or CAF external services.
- There is an intermediate layer that abstracts from the underlying Enterprise Service, so that only the used data is exposed to the composite application. The intermediate layer defines a service contract to the upper layers, and the implementation of these interfaces is called a service contract implementation layer.

- The upper layers of a composite application (Web Dynpro, VC, or BPM) only use these service contracts.

The architecture looks like this:

![Architecture Diagram]

With the service contract layer, it is possible to achieve decoupling from a concrete landscape, data type structure, communication protocol, etc. The service contract implementation layer can be implemented in different ways. First of all the composite application design has to be clear, if a complete decoupling has to be achieved. In that case SAP provides the following solutions:

- SAP NetWeaver BPM: the service contract implementation layer is implementing an interface by using BPM modeling capabilities to allow execution of service calls asynchronously, see SOA pattern [Asynchronous Write]. If an error happens during invocation of the Enterprise Service, a full error handling including user steps can be implemented.

- SAP Process Integration (PI): the implementation of the service contract implementation layer is not done via SAP NetWeaver CE; it is done via the process integration hub of SAP. The benefit is the same like in the case of SAP NetWeaver CE, the only difference is that the capabilities of SAP Process Integration can be used (like routing, ccBPM etc.).

- JMS: the service contract implementation layer contains a very small implementation of the service contract and only an event via JMS is raised that is processed on the Java server itself, so the message queue is used to decouple the functionality. The benefit is that only Java EE technologies are used.

The above options really decouple the execution of the Enterprise Service call from the basic composite application. If the asynchronous behavior is not a requirement, then the other technologies of SAP NetWeaver CE (CAF, Service Composition, EJB) to transform the delivered data of the Enterprise Services to the composite can be used. This solution has the drawback that a complete decoupling of the UI parts of the composite application is not achieved; it goes in the direction of tight coupling with all the disadvantages.

**Structure of the Service Contract Layer in a Composite Application**

The service contract and the service contract implementation layer have to be structured in a way that real decoupling is achieved. The structure of a decoupled composite application looks like this:
The entities of the composite application and the service contract definition are contained in one software component (SC) and the service contract implementation is contained in another SC. The contract and the implementation are in different SCs because this allows switching the implementations. These software components do not have any dependency to each other because they are loosely coupled and the communication between the composite application and the service contract implementation is only done via interfaces and configuration of the used services in the composite application.

**Concrete Implementation of Tight Coupling in Terms of Service Consumption**

Loose coupling is not always the best architecture. If flexibility is not required, performance indicates that an intermediate layer is not allowed, or dependencies to the backend systems are not an issue, SAP NetWeaver CE allows all frameworks (Web Dynpro, BPM, Visual Composer, etc.) to call services directly.

The architecture looks like this:

Example

The most commonly used implementation pattern in SAP NetWeaver CE is to implement an application service in CAF that is consuming one or many external services. The Java coding inside the application service is then responsible to map the data to the correct output data.

The intermediate CAF application service is used if one of the following abstractions is required for the composite application:

- Concrete landscape: The application services that are exposed by CAF can be deployed with the composite application. If the landscape changes at the customer’s side, however, other application services can be implemented using the same signature. Their implementation differs in such a way that the data is now retrieved from another system (or systems) within the customer landscape. Only a small configuration step is needed to activate the new application service for the composite.
• Data type structure: The Enterprise Services provide a specific data type structure and require these data structures. If the application is not interested in all data, it is ok to implement a CAF application service, define your own data types in CAF, and only expose the data structures of CAF.

• Communication protocol: CAF provides the functionality to call Web services and RFCs. If the application should be loosely coupled in the sense that the application should be able to run with Web services and get the data via similar RFCs later on, the recommended way is to implement a CAF application service and implement the switch inside the application service. Whether the data comes from an RFC or Web service is transparent for the upper layers of the application.

Web Dynpro and Visual Composer could either follow loose coupling or tight coupling architecture styles depending on the specific requirements, defined in the general considerations to use tight coupling or loose coupling.

... In Terms of CAF Services

Addressed issue: CAF is often used to implement business logic and access numerous services. This section describes how to structure CAF services into DCs to keep things manageable.

Business logic is usually implemented in CAF application services in a composite application. The functionality is put into a CAF project. The data is imported by external services and used inside the implementation of the application services. Bundling the CAF application services leads sometimes to CAF projects that contain many applications and even more external services. Unfortunately, this is a problem during development, because every change requires a regeneration of the external services (proxies) and the application services itself. In real-life applications the result is a long generation time.

Assume the composite application has a CAF project containing:

• n external services
• m node services
• k application services

In such situations, you should split the CAF project into several CAF projects. The basic idea behind that is that the application services and external services are put into different projects to reduce the generation time of the services (application, external).

1. Group the external services depending on e.g. their business semantics, usage, etc. For example, you may put the services that are frequently changed in the backend in one group.
2. Group the BO node services accordingly.
3. Group the application services accordingly.

The names of the groups are defined as following:

• E₁, …, Eᵢ (external services)
• B₁, …, Bᵢ (BO node services)
• A₁, …, Aᵢ (application services)

Now, one new CAF project for each of the groups has to be created. Between the new CAF projects dependencies can be created to use the entities (e.g. external services). The following rules have to be applied:

1. If an application service from the application Aᵢ uses an external service from the application Eᵢ then:
   o add a dependency from Aᵢ to Eᵢ;
   o add a dependency from the application service to the external service (or, more specifically, to the application service exposed from the external service via the automatic mapping).

2. If an application service from the application Aᵢ uses a BO node service from the application Bᵢ then:
   o add a dependency from Aᵢ to Bᵢ;
3. If an application service from the application $A_i$ uses application service from the application $A_j$ then:
   o add a dependency from $A_i$ to $A_j$;
   o add a dependency from the application service to the other one.

Here is an example:

![Diagram showing dependencies between application services and BO node services]

... In Terms of Task Handling in a Process

**Addressed issue:** a composite application usually contains process definitions and corresponding tasks. This section describes how to define the tasks.

In SAP NetWeaver BPM, the lifecycle of a task instance is bound to a surrounding process, even though it is possible to define a task as a standalone artifact in its own DC.

Under very specific conditions, the number of processes/tasks/DCs can be reduced. Given:

- Optional: multiple processes are modeled that contain a human activity as their single artifact (ignoring start/end event).
- The tasks that back the human activity all point to the same task UI and differ only in the occurrence of the process context.
- The task description can be constructed by using/interpreting the process context.

In a situation like this, implement only a single, generic process and generic tasks and let their work item description be based on the process context, which needs to be fed accordingly by the start event message.

We therefore also recommend investigating whether existing processes, tasks and task UIs can be refactored to make use of this approach.

As an example for a generic approach:

You have a set of different processes running in the backend (say ‘Process A’ and ‘Process B’). For each of the processes, a request for confirmation at a certain point should be requested from responsible users. Instead of modeling the processes ‘Confirmation for Step x (Process A)’ and ‘Confirmation for Step x (Process B)’, model a single process ‘Generic Confirmation’. Your start event message should carry all data that you require to make the necessary recipient determination and fill your fields in the Confirmation UI. Add a human activity to the process. Implement your Notification UI in Web Dynpro and fill all task specific information from the context. Connect your Web Dynpro UI as the task UI to the activity in your process. Pass all your task specific details from the start message to the human activity/task (thus Web Dynpro context) via data mapping.
Alternatively: If the confirmation task is still trivial and based on the same attributes of the different processes but requests different UIs, think of a gateway in the process to select the different task and thus the target UIs.

**… In Terms of Extensibility**

**Addressed issue:** the Extensibility Configuration Framework was introduced in NetWeaver CE 7.20 and the technology requires a specific structure. This section describes how the composite application has to be structured.

Extensibility is an important issue during development of a composite application. Extensibility is normally understood as the ability of an application to change its behavior without design modifying its design/code (only certain implementation parts are replaced). The structure of the composite application therefore stays, but new functionality can be plugged in.

There are many ways to design an application for extensibility purposes, see [8]. But SAP NetWeaver CE provides a specific framework for extensibility. The Extensibility Configuration Framework demonstrates an innovative conceptual approach, which allows different interface implementations of the application parts to be exchanged during the runtime of an application.

To do this, you first need to define *extension points* in your application. An extension point is a reference to a development object interface of a redirectable technology. The extension point allows exchanging, or actually redirecting to different interface implementations during the runtime of an application.

You can create extension points for the interfaces of the following development objects:

- Enterprise JavaBeans (EJBs)
- Composite Application Framework (CAF)
- Web Dynpro (WD)
- In the Adobe Interactive Form (AIF) use case, the extension point is not an interface, but a single form marked as extensible.

To create an extension point, there must be at least two development objects from one of the above types, as there needs to be a relation between them.

The extensibility configuration framework requires a specific structure of the application. The application has to be prepared to be extensible.

**CAF and EJB**

CAF and EJBs have to follow the same principle. Even if the EJB is only a place holder, the redirect from an existing implementation in an EJB can only happen in an EJB container. This means that an EJB that has to be redirected has to be invoked from another EJB.

The consequence of this limitation is that an EJB that is consumed by a Web application (Web Dynpro, servlet, etc.) has to be wrapped by another EJB, because invocation of the first EJB is performed by the Web container, not the EJB container of the Java EE engine.

**CAF and EJB**

NW BPM doesn’t support process extensibility via the Extensibility Configuration Framework as yet. In order to achieve extensibility at the sub-process level, invoke processes via their Web service interface through automated activities and not through referenced sub-processes.

**Product**

The extension components (components that replace functionality of the composite application) have to be put in a customer product. The original product does not contain the extensions; the extensions are put in the customer product that has a dependency to the original product.
... In Terms of External Libraries

**Addressed issue:** external libraries are usually important in Java development in SAP NetWeaver CE. This section describes how to use external libraries.

Using external libraries in the SAP component world is a complex task. This chapter explains the steps required to use an external library DC in your Java application using SAP NW CE 7.1. It is based on two blogs in SDN. Part I [3] gives you the necessary background, while part II [4] shows you how to do it in NWDS. Here, only conceptual information is provided.

External Libraries can be deployed on the AS Java in two different ways.

- **As a standalone library.** In this case, the application is deployed as an EAR archive, and other applications on the AS Java can use this external library, too.

- **As a bundled library.** The external library is packaged and deployed with your own application.

**Bundled case:** You have your own application consisting of 3 DCs. These are a Web module DC (Web DC), an EJB module (EJB DC) and the EAR project (EAR 1 DC). Your external library is packed in the extLib DC, which is packaged into the EAR 1 DC and also deployed with the EAR 1 DC.

The EAR 1 DC must have dependencies (build time) to the assembly public part of the EJB DC and the assembly public part of the Web DC. These dependencies are created automatically when you assign your Web and EJB modules to your EAR project. Public parts of type assembly can be packaged into the build result of the EAR file for deployment reasons. This means that the WAR archive (from the Web Module), the EJB.jar archive (from the EJB module) and the external library jar file are added to the EAR 1 DC.

If you want to **bundle** an external library with your EAR 1 DC, you first need to create a DC of type external library and create a dependency from your EAR 1 DC to the assembly public part of your external library DC. After you have defined this dependency, the external library files are automatically added to the EAR 1 archive and finally deployed with the application.

In order to use the external library classes, in your Web DC for example, you also need to add a dependency from your Web DC to the **compilation** public part of your external library DC.

**Standalone case:** You have your own application consisting of 3 DCs. These are a Web module DC (Web DC), an EJB module (EJB DC) and the EAR project (EAR 1 DC). Your external library is packed in the extLib DC, which is packaged in the corresponding EAR 2 DC. You can now deploy your external library (EAR 2 DC) like any other application. To use the external library from your application, you have to set a runtime dependency from your EAR 1 DC to the EAR 2 DC.
If you want to deploy your external library standalone, you have to create a external library DC and a new EAR 2 DC, which references the assembly public part of your external library DC. After you have defined this dependency, the external library files (jar files) are automatically added to the EAR 2 archive. The EAR 2 DC can now be deployed like a normal application.

If you want to use the external library in your own application (EAR 1 DC), you need to add a runtime dependency between your EAR 1 DC and the EAR 2 DC. To be able to use the library classes for example in your Web DC, you also have to add a dependency from your Web DC to the compilation public part of your external library DC.

In [4], an example to reuse libraries is described.

--- In Terms of Web Dynpro

Web Dynpro Faceless Component Structure for Web Services

Addressed issue: This section refines some of the existing guidelines on faceless components.

There are recommendations to structure Web Dynpro components into UI components and faceless components wrapping imported models for Web services. This information is available at [18] and [19].

Following the recommendation to use faceless components for models, two problems become apparent:

1. Faceless component structure: If the services that are used in Web Dynpro changed during development, the Web Dynpro model and the corresponding faceless component have to be adapted. So, a change in a service requires a long chain of adaptations in depending components, creating effort that should be reduced where possible.

2. Types: The faceless component exposes its data via context nodes so that a mapping is required. If the mapping is simplifying the complex structure of the original service, then the probability that the context nodes and values have a different type than the imported model is very high. Such a type mismatch can create problems at runtime, because the Web Dynpro runtime checks the data types only at runtime. This could lead to exceptions.

There are different proposals for each of the problems:

1. The faceless component that wraps a Web Dynpro model should be put into a DC and the component should be exposed via a public part. Only the interface of the faceless component should be used. It should be avoided that the same service is imported many times in one composite application, because using the same services including the same types, but importing it many times in a composite applications leads to class loading problems. The type is imported in every model and the corresponding Java classes are instantiated at runtime and should be identical, but they are not, because they belong to different archives.

2. The values typed by specific data types are checked during runtime. There is a way to avoid the check operation (see note 1310216), but this requires a new design of the service.
Web Dynpro View and Component Structure

Addressed issue: This section adds additional WD structuring recommendations on top of general DC structuring recommendations.

Web Dynpro is a powerful programming model based on the MVC paradigm. Therefore, there are many possibilities to use the entities of Web Dynpro. The structuring of the entities can be done in different ways based on the requirements at hand.

If the structure of a Web Dynpro entity is not fine-granular and structured enough, there are problems to do team development as only one developer can work at a time with a given entity. Therefore, it is a good practice to structure the entities based on the requirements regarding team development, business purposes, etc (see chapter [… In Terms of SCs and DCs] for details).

Each programming model entity within Web Dynpro fulfills a dedicated purpose. The following entities are particularly important for this discussion:

- Component: a Web Dynpro component is the reusable entity of the programming model. It has a clearly defined interface. A component has a corresponding controller.
- Custom controller: a custom controller contains the logic and bundles the data for a dedicated purpose.
- View: a view is the entity to display and to render the user interface. A view has a corresponding controller.

Based on these definitions, the following structure recommendations can be derived:

1. If the user interface is generally reusable, there should be a development component (DC) that contains a specific Web Dynpro component for the UI. The component interface is put into the public part of the DC.
2. If the user interface or the functionality of a Web Dynpro component is reusable inside a DC, then the user interface or the functionality is put into a dedicated Web Dynpro component.
3. The functionality inside Web Dynpro should be grouped according to business reasons (user interface or logic). There are different ways to structure the content:
   a. If the functionality of the user interface can be reused and is of general interest, then it is the best way to define a Web Dynpro component for that. A Web Dynpro component has a clear interface and is reusable.
   b. If the user interface should be developed by many developers, the user interface should be split into different components (see above). If the user interfaces are too small for a dedicated Web Dynpro component, it is a good design paradigm to define views in a way that the developers can work independently. The data transfer is done by a custom controller that holds the data that is used in each of the views.

... In Terms of Business Rules

Invocation of Business Rules

Addressed issue: rules can be applied in a process either using a ruleset function or Web service. This section compares these options and describes when to use which.

There are two ways to define and use rules in a process. One is modeling the rules as part of the Process Composer DC itself as ruleset functions and using them in mappings. The other way is to model the rules as a ruleset in a Rules Composer DC and to expose the ruleset as a Web service and call the service from an automated activity.
<table>
<thead>
<tr>
<th>Reusability</th>
<th>Within the Process Composer DC</th>
<th>Across different Process Composer DCs and even in other CE technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restricted features</td>
<td>(a) Apply ruleset N/A</td>
<td>No restrictions</td>
</tr>
<tr>
<td></td>
<td>(b) Reusable Rulesets N/A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(c) Cannot expose ruleset function as Web service</td>
<td></td>
</tr>
<tr>
<td>Signature (Input / Output)</td>
<td>Fixed</td>
<td>Fixed, but services with different signatures can be created using service adaptations</td>
</tr>
<tr>
<td>Performance</td>
<td>Slightly faster</td>
<td>Slightly slower</td>
</tr>
</tbody>
</table>

- **Life Cycle**: the life cycle of a ruleset function is the same as that of the Process Composer DC in which it’s defined. Rules exposed as Web services will have the same life cycle as that of the Rules Composer DC containing the rules. Rules defined in Process Composer DC and Rules Composer DC can both be modified at runtime using Rules Manager.

- **Reusability**: a ruleset function is only reusable within the Process Composer DC where it was defined, whereas rules exposed as Web services can be reused in other Process Composer DCs and other CE technologies (CAF, WebDynpro, etc.) as well.

- **Restricted features**: There are some features which are not available for rules defined in a Process Composer DC. For example *Apply Ruleset* and *Reusable Rulesets* are not supported. These features are available for rules defined in Rules Composer DCs only.

*The reusable rulesets feature is available from SAP NetWeaver CE 7.3 onwards.*

- **Signature**: The ruleset function signature is fixed. Although the signature of a ruleset exposed as Web service is also fixed, it's possible to create multiple Web services with different signatures for same ruleset using service adaptations in Service Composer.

- **Performance**: Runtime performance of ruleset functions is slightly better than that of rules exposed as Web services. To better illustrate this, the performance of a ruleset function and a ruleset exposed as Web service was measured using a process using the ruleset function and another process using the WS-enabled ruleset. Rules were the same in both cases. The time taken for the process to complete was measured with varying size of the initial XML payload used to start the process. The following table contains the test results:

<table>
<thead>
<tr>
<th>Payload Size</th>
<th>Process Using Ruleset Function</th>
<th>Process Using Ruleset Exposed as Web Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 MB</td>
<td>5.25 seconds</td>
<td>5.75 seconds</td>
</tr>
<tr>
<td>2 MB</td>
<td>10.5 seconds</td>
<td>12.25 seconds</td>
</tr>
<tr>
<td>4 MB</td>
<td>21.5 seconds</td>
<td>25 seconds</td>
</tr>
</tbody>
</table>

On a broad level it’s recommended to use ruleset function when reusability of rules is limited due to the localized nature of rules and tight coupling between process and rules is required / desired. Other factors to be considered are rules defined in a ruleset function cannot be further organized and separated into rulesets and reusable rulesets.
Decision Table Evaluation

Addressed issue: Decision Tables capture rules in a tabular format which is easy to understand and intuitive for business users. This section describes how decision tables are evaluated, how different properties of a decision table affect its runtime behavior, and how to organize large decision tables.

(a) The evaluation starts with condition column 1. The order of evaluation in a condition column is from top to bottom. Once a condition value in the condition column is satisfied, the search is limited to that span of condition values in next condition column.

(b) This narrowing of search in each subsequent condition column continues till all condition values in a logical condition row are satisfied. Actions of the satisfied logical condition row are fired.

Evaluation of Decision Table **discount_calculation** which calculates discount percentage:

<table>
<thead>
<tr>
<th>Customer code = 0012</th>
<th>Item price = 250</th>
<th>Item Quantity = 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>purchaseOrder/customerCode</td>
<td>Price of Item</td>
<td>Item Quantity</td>
</tr>
<tr>
<td>&lt; 100</td>
<td>&lt; 10</td>
<td>0</td>
</tr>
<tr>
<td>Between 100 and 1000</td>
<td>&gt;= 10 and &lt;= 50</td>
<td>1.5</td>
</tr>
<tr>
<td>Between 1200 and 1500</td>
<td>&gt;= 70 and &lt;= 80</td>
<td>2.0</td>
</tr>
<tr>
<td>&gt; 1000</td>
<td>&gt; 80</td>
<td>4.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer code = 0013</th>
<th>Item price = 2000</th>
<th>Item Quantity = 75</th>
</tr>
</thead>
<tbody>
<tr>
<td>purchaseOrder/customerCode</td>
<td>Price of Item</td>
<td>Item Quantity</td>
</tr>
<tr>
<td>&lt; 100</td>
<td>&lt; 10</td>
<td>0.5</td>
</tr>
<tr>
<td>Between 100 and 1000</td>
<td>&gt;= 10 and &lt;= 50</td>
<td>1.5</td>
</tr>
<tr>
<td>Between 1200 and 1500</td>
<td>&gt;= 70 and &lt;= 80</td>
<td>2.0</td>
</tr>
<tr>
<td>&gt; 1000</td>
<td>&gt; 80</td>
<td>4.9</td>
</tr>
</tbody>
</table>

The evaluation order of a Decision Table is from top to bottom. Hence it's recommended to put logical condition rows which satisfy quite often at the top of the Decision Table and logical condition rows which satisfy rarely at the bottom of the Decision Table to achieve better performance. This difference in performance might be negligible for small Decision Tables evaluated with small number of inputs. But it becomes quite significant for large Decision Tables evaluated with large number of inputs.

Tests were conducted with a simple Decision Table containing one condition and one action with 1000 logical condition rows on SAP NetWeaver CE 7.2. An input data (XML) of 2 MB size was used. The following results were achieved:

<table>
<thead>
<tr>
<th>Satisfied Logical Condition Row</th>
<th>Time Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.86 seconds</td>
</tr>
<tr>
<td>500</td>
<td>21.76 seconds</td>
</tr>
<tr>
<td>1000</td>
<td>39.48 seconds</td>
</tr>
</tbody>
</table>

* Test results shown here are indicative only and should not be used in absolute terms as Decision Table evaluation depends upon several other factors.

Properties of Decision Tables

Addressed issue: a Decision Table has two important boolean properties - **Rows are mutually exclusive** and **Return multiple matches**. These two properties control the runtime behavior of a Decision Table. This section explains their semantics and usage scenarios.
**Rows are mutually exclusive:** this property should be set to true if all logical condition rows in a decision table are mutually exclusive. Otherwise it should be set to false.

In above mentioned Decision Table `discount_calculation`, no condition value `<10, >=10 and <=60` within the span of condition value `{0012, Between 1200 and 1500}` satisfies the input `{customer code = 0012, Item price = 1250, quantity = 80}`.

If the `Rows are mutually exclusive` property is set to true, no further evaluation of the Decision Table happens although there is a logical condition row `{0012, >1000, >80}` which satisfies the input. Because of this no actions of are fired.

If the `Rows are mutually exclusive` property is set to false for the same Decision Table and the Decision Table is evaluated with same input i.e `{customer code = 0012, Item price = 1250, quantity = 80}` the discount percentage is set to 4.0 as a result of logical condition row `{0012, >1000, >=10 and <=80}` getting satisfied.

'**Rows are mutually exclusive= true**':

![Decision Table diagram showing mutually exclusive conditions](image)

'**Rows are mutually exclusive= false**':

![Decision Table diagram showing non-mutually exclusive conditions](image)

'**Return multiple matches**': this property should be set to true if there are multiple logical condition rows which satisfy a given input and actions of all such logical condition rows are to be fired. The actions of all
satisfied rows are fired in the same order they are defined. This property should be set to false if there is only one logical condition row which satisfies a given input or if there are multiple but only actions of the first one are to be fired.

In the above mentioned Decision Table \textit{discount\_calculation}, there are multiple logical condition rows (highlighted with a line) which satisfy input \{\textit{customer code = 0012, Item price = 1250, quantity = 10}\}.

If the \textit{Return multiple matches} property is set to false, the discount percentage is set to 3.5.

If the \textit{Return multiple matches} property is set to true for the same Decision Table and the Decision Table is evaluated with the same input, i.e. \{\textit{customer code = 0012, Item price = 1250, quantity = 10}\}, the \textit{discount percentage} is set to 4.5. Although there are multiple logical condition rows which satisfy the given input, the actions of the last one, i.e. \{\textit{Like 00*, > 1000, >=10 and <=80}\}, override previously fired actions.

\textit{‘Return multiple matches = false’}:

\textit{‘Return multiple matches = true’}:
It's very important to understand these two properties of Decision Tables and to set them accordingly. Results of setting incorrect values for these properties can range from performance degradation to incorrect behavior.

**Decision Table Size**

**Addressed issue:** Decision Tables can grow large rather quickly. This section gives recommendations on how to structure Decision Tables.

100,000 cell entries (number of columns X number of logical condition rows) is the recommended maximum size for a Decision Table. Beyond that, it becomes difficult for tools to manage the Decision Table and also difficult for humans to comprehend it. In such cases, it's recommended to split the Decision Table into multiple Decision Tables and link them through the dynamic Decision Table functionality.

The runtime performance of evaluating large Decision Table and smaller broken Decision Tables will not differ much. So the decision to break large Decision Tables should be based on the design time and other factors like import/export of Decision Table from/to excel spreadsheet and editing in Rules Manager.

**Rulesets Exposed as Web Services**

**Addressed issue:** a ruleset can be exposed as a Web service that can be consumed like any other Web service. This section describes how they can be used in certain scenarios.

**Request / Response Wrappers**

When a ruleset is exposed as a Web service, request and response wrappers are created around input and output elements. These wrappers are created to handle multiple inputs and outputs and are always created. This might not be desirable when there is only one element present in input and output. In such cases, the ruleset exposed as a Web service can be adapted not to contain wrappers around input and output using a service adaptation as illustrated below:

![Service Adaptation Diagram](image)

**Multiple Web Services for a Ruleset**

Currently only one Web service is permitted per one ruleset and it's not possible to create multiple Web services with different signatures for same ruleset. But this can be achieved by first exposing the ruleset as Web service with all the elements used in that ruleset as input and output. This Web service can then be used to create multiple Web services with different signatures using service adaptation as illustrated below.
Rules Engine Configuration Properties

Addressed issue: there are certain configuration properties which affect performance and behavior of the Rules Engine. This section describes their behavior, how they affect performance, and what the preferred values are.

The following configuration properties are available to configure the Rules Engine,

- Refresh Time (integer)
- Maximum Cache Size (integer)
- Business Log Level (integer)

These properties are available as EJB Environment Entries in the EJB module tc/brms/engine/ejb in J2EE application tc/brms/engine/app.

Refresh Time

Rules Engine caches networks (Rete, Flow) created when rulesets are invoked. This caching of networks improves the Rule Engine performance when the same ruleset is invoked next time. Since the ruleset can change in repository after its network is cached, Rules Engine requires checking whether the ruleset is modified in the repository. This requires checking specific fields in rules runtime DB tables. Since this is a DB operation and some overhead is involved, checking of whether ruleset is modified or not is controlled through Refresh Time property. When a ruleset is invoked and if the network is cached, Rules Engine checks the repository for any changes in ruleset if the time elapsed since last time the check is done is more than the Refresh Time. Otherwise Rules Engine continues to use the cached network.

Recommendation: If changes to rules need to be reflected immediately in the next invocation of the ruleset, this value must be set to zero. Also value of this property should be checked if changes to rules are not getting reflected immediately.

Maximum Cache Size

Rules Engine caches networks (Rete, Flow) created for different rulesets. The value of Maximum Cache Size determines how many such different rulesets networks can be cached. If the different rulesets invoked exceed this value, the rulesets networks are removed from the cache using the Least Recently Used (LRU) algorithm. There could be multiple instances of Rules Engine as it’s a stateless session bean and the EJB container pools EJBs to handle concurrent requests to the Rules Engine. So the total memory occupied in caching networks is sum of the cache memories in different instances of Rules Engine.

Recommendation: A value equal to or slightly more than the total number of rulesets in all Rules Composer DCs and Process Composer DCs should be set for this property to achieve better performance. If a too large value is set for this property, this can lead to memory issues.
**Business Log Level**

Rules Engine logs the execution details in form of business logs using the BPM Business Log. But the amount of business logging done by Rules Engine is controlled by this property. Setting high *Business Log Level* will result in degraded Rules Engine performance.

**Recommendation:** It’s recommended to use a low Business Log Level unless it’s absolutely required to have high level of business logging.

---

**Choice of Composition Technologies**

SAP NetWeaver CE provides frameworks for different purposes. There are frameworks for business logic implementation, user interface modeling, and process modeling. The functionality of these frameworks sometimes overlaps, and you need to decide when to use which framework. The following explanations will give some guidance.

### ... Regarding UI Flow (BPM and Web Dynpro)

**Addressed issue:** the various steps of a user interface can be implemented in different ways. In particular, Web Dynpro and BPM can be used to model the user interface steps. This section describes how to decide how to model the steps.

In general, both technologies, Web Dynpro and BPM, play well together for the realization of business processes or composites. It depends on the concrete requirements of a business process to decide on how to wire them together or whether to use them exclusively. As already described, business processes can come in a plethora of different occurrences. In this chapter, we will focus just on human interaction, because this is where SAP NetWeaver BPM and Web Dynpro are complementary.

The two technical offerings are best described by the domains that they belong to. SAP NetWeaver BPM allows you to model and execute workflows (generally it is a process orchestration engine) while Web Dynpro (for Java) is a UI technology with screen-flow capabilities. As indicated by the suffix ‘flow’, both technologies are able to deliver ‘one-to-many’ business activities to human users. While Web Dynpro screens-flows are about the interaction with a single user, BPM focuses for a more global approach where a single flow model can involve multiple users on multiple activities.

The following discussion assumes that you leverage BPMN as the language to describe your business process (and discuss it with the line of business (LOB))

**Human Activities**

When implementing a business process with concrete technology, it is important to understand the difference between a human activity in a pure descriptive BPMN model and their realization in a system. At present, SAP NetWeaver BPM only allows implementation of human activities as tasks. A task is a very explicit means. It will create a work item in an inbox (here UWL) which has to be picked up by the corresponding user to be executed. There is currently no option to use BPMN models in SAP NetWeaver BPM to express that individual human activities shall be grouped and executed as a screen-flow for a certain task (although this might change in the future, especially for consecutive activities). This provides a hint about the typical division of labor between the technologies. While SAP NetWeaver BPM executes an overall business process flow to instantiate tasks, Web Dynpro is to be leveraged when interacting with a user on an individual task.

As a result, some of your human activities of the descriptive model will not make it to the executable BPMN model in SAP NetWeaver BPM. Others will not show up because you implement them as Views in Web Dynpro.

**Special Composites/Processes**

When transforming a descriptive BPMN model into an executable model, you might end up with the situation that all activities have to be executed by a sole user. Here, the only remaining question is how the user will execute this process: either as a task or as a self-initiated activity. If it should be a task, a process with a single human activity has to be modeled in BPM, and all the logic goes into Web Dynpro and is linked to the
BPM activity as a task UI. If the latter applies, you will implement a Web Dynpro application only and share the link with your users (via Portal technologies, for example).

**Initiation of Processes**

At present, the processes in SAP NetWeaver start with an event message. There are three options for this:

- Call the process directly via Web service technology
- Connect to the process indirectly via the event (message) infrastructure
- Use the ‘SAP NetWeaver Administrator’ functionality

If you want your processes to start with a user initiated UI activity, there is currently no way to model this in SAP NetWeaver BPM. The proposal is to implement a Web Dynpro application and call the Web service endpoint of the process directly when required. In your modeled BPM process, add a comment referring to the Web Dynpro UI which is intended to initiate this process.

---

**Regarding UI Modeling (Web Dynpro and VC)**

**Addressed issue:** the SAP NetWeaver CE contains different UI programming models. This section describes when to use which.

SAP NetWeaver CE contains a number of UI programming models that require a clear separation. The frameworks explained here are Web Dynpro Java with NWDS tools and Visual Composer with Web Dynpro Java runtime (WD4VC).

First of all, the frameworks will be explained from a technical point of view and how they are related. Web Dynpro is a UI framework that runs in the Web container of the Java EE server. It provides its own component model for reuse, which means that the Web Dynpro component is a very crucial entity in the whole programming model. The Web Dynpro runtime, running in the Web container, can instantiate and execute an arbitrary number of components. The connection to Visual Composer is such that the runtime to execute Visual Composer models is implemented as Web Dynpro components. In principle, all the features that Visual Composer models can make use of at runtime are therefore available in Web Dynpro at the same time, but if something is available in Web Dynpro, this does not mean it is automatically available in Visual Composer; it has to be implemented in the components.

The Web Dynpro development environment is Eclipse based, and a large number of tools for the different entities of the programming model are available. For Visual Composer, there is a browser based modeling environment, implemented in JavaScript, HTML, and the SVG plug-in from Adobe. As a result of implementations in two different environments, they do not reuse common frameworks. The Visual Composer development environment might therefore provide more features in the area of developer productivity than Web Dynpro itself. The Visual Composer development environment is integrated in Eclipse using the HTML editor integration in Eclipse, meaning that the Visual Composer models can be edited in NetWeaver Developer Studio, too. Using this integration therefore makes it possible to navigate and edit Visual Composer models, and some basic frameworks that are available in NetWeaver Developer Studio only can be used by Visual Composer. The most prominent example of a framework that is only available in Visual Composer in Eclipse is the mass configuration support via Service Groups. These entities cannot be created in the browser-only environment, but only in Visual Composer in Eclipse.

Having said this, the easiest way to explain the differences is to summarize it in a table and provide more details afterwards:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Web Dynpro</th>
<th>Visual Composer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adobe Interactive Forms support</td>
<td>Basic support of forms by providing a UI element that can host an interactive form</td>
<td>No support of Adobe Interactive Forms</td>
</tr>
<tr>
<td>Stateless applications and asynchronous UI parts</td>
<td>Available</td>
<td>Only stateful applications</td>
</tr>
<tr>
<td>Analytics</td>
<td>Implementation dependent, no direct</td>
<td>Visual Composer BI Kit to access</td>
</tr>
</tbody>
</table>
### Extensibility support

<table>
<thead>
<tr>
<th>Functionality</th>
<th>SAP BusinessObjects Design Studio (BODS)</th>
<th>Web IDE</th>
<th>SAP HANA (HANA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data federation</td>
<td>Directly integrates data from various sources</td>
<td>No direct integration</td>
<td>Integrates data through BODS</td>
</tr>
<tr>
<td>Analytics</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Notification Management</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Scripting</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Deployment</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Expressions

- **Implementation-specific, no declarative tool support**
- Properties can take expressions as values, similar to expression language in Excel

### Implementation by native language

- **Web Dynpro provides a Java API to allow Java coding for development**
- Implementation in Visual Composer models is not supported

### Developer productivity

- **Model driven tools integrated with implementation tools**
- Visual Composer is designed to allow fast development of modeled UIs ➔ the whole design time is tailored for a small learning curve

---

- **Adobe Interactive Forms**: With Web Dynpro, a UI element can be put to a Web Dynpro view and the data transferred from the Web Dynpro context to the Adobe Interactive Form, thus allowing direct context binding. The form is persisted in a specific folder of the Web Dynpro project and can be edited by the dedicated Adobe Template Designer.

- **Stateless applications and asynchronous UI part**: Stateless Web Dynpro Java applications keep all information needed to approve a user’s request on the client. This information is passed to the server on each roundtrip. This allows the server to release resources during the user think times and fits perfectly with scenarios where a large number of users have either long “think times” (one roundtrip in half an hour say) or where the application they use only needs a small amount of information for the interaction step (so startup is inexpensive). Stateless applications do not suit applications which need a lot of information to work properly.

  Web Dynpro Java applications can embed other Web Dynpro Java applications. The user sees a seamless UI and can interact with both of them. However, the embedded application can run asynchronously to the embedding application, meaning that a response can be sent to the client while the embedded application is still running. The screen area of the embedded application can be filled with an intermediate result by the embedded application, and the client is instructed to look for content updates after a certain amount of time has elapsed. Once the embedded application has finished processing a user roundtrip, and the update has been transported to the client, the corresponding UI area is unlocked, allowing for user interaction.

- **Analytics**: The BI capabilities of NetWeaver can be accessed in Visual Composer by a specific extension to Visual Composer - the BI kit. This kit allows the connection to BI systems to display data. It is a very natural type of data integration, because the handling of data is completely integrated into the Visual Composer modeling environment. BI functionality is only available in the browser-based solution of Visual Composer.

- **Extensibility**: In Web Dynpro, it is possible to mark components that can be replaced by other component implementations, fulfilling the same interface. The replacement of the original components does not require any modification of the existing application.

- **Expressions**: In Visual Composer, expressions can be used as values for nearly all properties. By doing this, Visual Composer provides an easy way to make a lot of changes without detailed knowledge of a programming language. The expression language is similar to the expression language available in Excel for formulas and the like. To achieve similar behavior in Web Dynpro, Java coding is required.

- **Implementation by native language**: Web Dynpro is based on the Web container implementation and provides a Java API, so nearly all entities can be edited by Java calls. This gives the user complete freedom to implement all UI applications.

- **Developer productivity**: The area of developer productivity is very broad. To explain the difference between Web Dynpro tools and Visual Composer, it helps to explain the starting points of the frameworks. Web Dynpro was started as a complete programming model to enable easy development of UIs for Enterprise applications. Visual Composer started as a composition tool for...
business user to create small UIs for reporting. Based on the different focus groups, Web Dynpro for developers, Visual Composer for business user, the focus was on different areas.

Web Dynpro provides all entities and APIs to implement a powerful application and support the developer in implementing complex applications. Over the years, Web Dynpro improved in the area of development tools, but so far has not achieved the same level of simplicity as Visual Composer for simple UIs. Nevertheless, Web Dynpro provides developers with excellent support when developing complex UIs.

**Combination of both paradigms**

Unfortunately, Visual Composer models cannot be transferred to Web Dynpro components. This makes it impossible to start with the better developer productivity and go to Web Dynpro at a later point in time. During application design, the developer has to think which user interfaces are simple enough to be modeled in Visual Composer and what is too complex to be modeled with Visual Composer. The additional features have already been listed and described above.

The two user interface programming models are not completely isolated. Simple UIs can be modeled in Visual Composer, while the complex parts are developed in Web Dynpro and integrated via black box integration into Visual Composer.

### Regarding Data Transformation (CAF, BPM, SA)

**Addressed issue:** SAP NetWeaver CE contains different programming models for data transformation. This section describes when to use which.

**Decoupling**

If a composite application developer has to design an application, s/he first of all has to think about tight or loose coupling. In this area, there are a number of recommendations for the specific frameworks available for data transformation.

- **Tight coupling style** is recommended when:
  - **Single-place consumption** of a particular service is needed in the scope of a composite application. In cases like this, we recommend importing and transforming (adapted/simplified/mapped) services or using them directly in WD/VC/BPM.
  - **High performance** is needed when consuming a service. Mediation steps with CAF or Service Composer are considered overhead compared to direct service consumption.

- **Loose coupling style** is recommended when:
  - **Service adaptation reuse** is needed in multiple places (screens, processes, etc.) in a composite application. We then recommend mediating/wrapping this service in CAF or Service Composer. The service is consumed, transformed (adapted/simplified/mapped) and then exposed using the CAF or Service Composer, thus making the transformation logic re-usable. We recommend using Service Composer when the transformation logic complexity is low to moderate and can be modeled with simple flow diagrams and structure mappings. We do not recommend implementing business logic with Service Composer, since conditional and error handling behavior modeling is not supported yet (7.20). We recommend using CAF when the transformation logic is too complex and implementation is easier with Java coding. We also recommend using CAF up to SAP NetWeaver CE 7.20 if high performance is a mist (performance of Service Composer has been greatly improved to match CAF in version 7.30 onwards). CAF is also recommended in cases where consumption of a backend services alone is not sufficient, and additional data (specific to the composite) has to be managed, in a local database for example.
  - **Service composition reuse** is needed in multiple places in a composite. We then recommend importing, transforming (adapting/simplifying/mapping) or using consumed services as-is and then
composing them (result is exposed as a single service) in CAF or Service Composer, thus making the composition logic reusable.

- **Backend abstraction layer** pattern is needed to allow for backend system decoupling at runtime.

  We then recommend implementing the platform independent service interface, backend service adaptations and the switching configuration mechanism [1] with CAF.

**Feature Comparison**

CAF and Service Composer both provide data transformation capabilities. The differences are in the specific details of the frameworks. The following section provides a basic description for the developer of a composite application and support him/her in the decision process:

<table>
<thead>
<tr>
<th>Capability / Tool</th>
<th>CAF</th>
<th>Service Composer</th>
<th>Java EE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service consumption (WS, RFC)</td>
<td>Modeled &amp; generated to leverage JAX-WS &amp; JCo static proxies for optimum performance, No service mocking, No Service Interface re-import</td>
<td>Modeled &amp; interpreted; service mocking, Service Interface reimport with delta detection, flexible protocol switching</td>
<td>JAX-WS, JAXB, JCo</td>
</tr>
<tr>
<td>Type Re-use &amp; collisions handling</td>
<td>Conflicting type definitions cannot be imported (first wins)</td>
<td>Conflicting type definitions are interactively imported</td>
<td>n/a</td>
</tr>
<tr>
<td>Service simplification</td>
<td>n/a (modeled manually)</td>
<td>Modeled (simplification wizard SI) &amp; automatically derived mappings &amp; service consumption)</td>
<td>n/a (coded manually)</td>
</tr>
<tr>
<td>Flow</td>
<td>Coded</td>
<td>Modeled &amp; interpreted</td>
<td>Coded</td>
</tr>
<tr>
<td>Data Transformation Mapping</td>
<td>Modeled &amp; generated to static code, one-to-one, constant mapping</td>
<td>Modeled &amp; interpreted, emulation, expression language, one-to-one, one-to-many, many-to-many, constant mapping, mapping functions, iterations (7.30), collection handling, type conversion, type casting (7.30), built-in functions, custom functions, function nesting</td>
<td>Coded</td>
</tr>
<tr>
<td>Service Provisioning</td>
<td>Modeled &amp; generated to leverage JAX-WS &amp; EJB</td>
<td>Modeled &amp; generated to leverage JAX-WS &amp; EJB</td>
<td>JAX-WS</td>
</tr>
<tr>
<td>Persistency</td>
<td>Remote persistence, entity-level permissions, instance level permissions, language dependent attributes, generated JPA entities, multiple (bulk) operations over BO nodes, SAP NetWeaver BW integration, business data transport, modeling of custom findBy operations, service browser for runtime testing, graphical modeler (7.11)</td>
<td>n/a</td>
<td>JPA</td>
</tr>
<tr>
<td>Custom coding in Java</td>
<td>Controlled</td>
<td>Limited to mapping functions</td>
<td>Unlimited</td>
</tr>
<tr>
<td>Write operations</td>
<td>Logic is implemented and therefore the execution of many service calls in a chain can be implemented in a way that errors can be handled</td>
<td>If there are many write operations in a model, then it is difficult to do a good-enough error handling, because there is no way to trigger user interaction. Only if the error is exposed and some implementation outside of the model reacts to it</td>
<td>Logic is implemented and therefore error handling is possible to any degree</td>
</tr>
</tbody>
</table>

CAF and Service Composer are both model-driven (MDA) tools for service development, but they provide different design time and runtime capabilities, resulting in corresponding benefits. Depending on the particular use case it might be better to use one or the other technology.

- **Service consumption**: In CAF, WSDL artifacts can be imported to consume Web services and JCo models to consume RFCs/BAPIs. From the consumed services models, it then generates JAX-WS and RFC static proxies, resulting in improved consumption performance. Since communication technology specific proxies are used, it is not possible to switch the type (WS/RFC) of the consumed service once the service interface has been imported. Service Composer uses WSDL artifacts to
describe imported services for both WS and RFC types. Instead of generating static proxies (as in CAF), WSDL description is interpreted at runtime by SCA runtime and corresponding SDO Data Access Services (for WS or RFC), which leads to lower performance compared to static proxies, but makes it possible to switch between WS and RFC types of consumed services. Easy service mocking can also be applied. Service Composer provides service interface re-import with delta detection capability, thus preserving the existing service flow and data transformation mapping models.

- **Type reuse & collision handling:** Both CAF and Service Composer provide type reuse functionality, but they offer different type conflict resolution approaches. In CAF, conflicting type definitions cannot be imported, and the first type definition wins. In Service Composer, conflicting type definitions can be imported. Users are prompted to pick one of the conflicting type versions, and - if the newly-imported version is selected - all of the occurrences of the conflicting version in the existing (previously imported) WSDL and XSD documents are replaced with the new one.

- **Service simplification:** Service Composer offers a wizard to create a new (simplified) service from the existing service interface. The new service interface definition starts by taking the service interface of a service to be consumed and simplified and then selectively removing parts of the input and output data types. In the end, this results in a completely implemented simplified service delegate to the original service. It contains a new (simplified) service interface, import of the original service and corresponding data transformation mappings. In CAF, all steps have to be performed manually: modeling the new (simplified) service interface, importing the original service, defining mappings.

- **Service execution flow:** Service Composer offers a graphical modeling tool to define service execution flows using a minimal subset of BPMN, where only unconditional transitions are supported. In CAF, execution flows cannot be modeled but are coded in Java using dedicated custom code sections, thus allowing more sophisticated flows – conditions, error handling, etc.

- **Data Transformation mappings:** CAF provides a graphical tool for modeling data transformation mappings, which then leads to the generation of Java code that implements the actual transformation. This allows for higher runtime performance. Only one-to-one and constant mappings are supported in CAF. Service Composer offers easy graphical modeling for much more sophisticated mapping definitions. In addition to the one-to-one, one-to-many, many-to-many and constant mapping styles there are capabilities like mapping functions, built-in function libraries, custom functions, function nesting, iterations, collection addressing, type conversion, type casting (SAP NetWeaver CE 7.30) and evaluation at design-time. Service Composer mappings are dynamically interpreted at runtime, implying poorer performance than static mappings in CAF.

- **Service provisioning:** CAF services are generated in both Web service and EJB form using JAX-WS and JAXB. As Apart from JAX-WS & JAXB service implementations, Service Composer generates SDO-based EJBS which are exposed as Web services through the SCA runtime.

- **Data Persistency:** Service Composer does not offer data persistency capabilities, since it is focused on service adaptation (like simplification) and composition only. In CAF, graphical and form based modeling can be used to define models of the persisted data types, which are then used to generate JPA entities. There are also services on top of the JPA entities, like data type and data instance level permissions, language-dependent attributes, bulk operations (7.30), SAP NetWeaver BW integration (SAP BW pulls data from CAF BO Nodes) and business data transport. CAF also provides modeling of “findBy” operations, which are less powerful than JPAQL but much easier to define. There is also the “ServiceBrowser” Web tool for runtime testing of services developed with CAF.
- **Custom coding**: The custom coding in Service Composer is limited to the area of custom data mapping functions, where functions can be implemented with EJB, while CAF provides much more controlled opportunities for custom code. In CAF, custom coding can be created for declared Application Service operations, and can override any operation in BO Node and External Services. Any code that is not covered by any of the models in the “ejbmodule” folder can also be added.

- **Write operations**: The Service Composer is a model-driven tool (like explained before), so all functionality has to be modeled. Unfortunately, this would even be true for the error handling, so there are definitely some limitations for the error handling in comparison to BPM (where human activities are possible to do error handling and error states to start compensating successful service calls, see [Error Resolution: Compensation]). These limitations make it difficult to handle many service calls in a chain in the Service Composer, therefore it is usually better to handle write operations in the frameworks that allow custom coding and handling of all cases (CAF, JEE).

CAF provides better overall performance at runtime, while Service Composer provides improved usability. Below, you can find some example scenario measurements comparing Service Composer and CAF in terms of runtime performance.

To illustrate this fact, the following table contains a number of performance indicators. The measurement was made on one PC, so the indicators are comparable to the others, although there is no guarantee that every PC with the same performance achieves the same.

Technical scenario definition: Average payload size of composed service: 12kB, local Java services, protocol HTTP.

<table>
<thead>
<tr>
<th>Technology</th>
<th>Test Duration</th>
<th>Number of Transactions</th>
<th>Average Response Time</th>
<th>Java AS CPU</th>
<th>Memory per Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td>~2hours</td>
<td>126296</td>
<td>113ms</td>
<td>60%</td>
<td>13.2MB</td>
</tr>
<tr>
<td>CAF</td>
<td>~2hours</td>
<td>127062</td>
<td>45ms</td>
<td>39%</td>
<td>5.9MB</td>
</tr>
</tbody>
</table>

Technical scenario definition: average payload size of composed service: 12kB, remote Java services, protocol HTTP.

<table>
<thead>
<tr>
<th>Technology</th>
<th>Test Duration</th>
<th>Number of Transactions</th>
<th>Average Response Time</th>
<th>Java AS CPU</th>
<th>Memory per Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td>~2hours</td>
<td>125845</td>
<td>117ms</td>
<td>57%</td>
<td>12.5MB</td>
</tr>
<tr>
<td>CAF</td>
<td>~2hours</td>
<td>126503</td>
<td>70ms</td>
<td>35%</td>
<td>5.2MB</td>
</tr>
</tbody>
</table>

**Type Conflicts in the Frameworks**

Types are used in service consumption in SAP NetWeaver CE. All frameworks are able to consume Web services and XSD data type definitions, but there are some limitations, if SAP Enterprise Services are consumed.

According to the XML Schema specification any kind of a XSD entity with global scope (type definition, element declaration, attribute declaration, etc.) is uniquely defined by its QName, i.e. the ordered pair of target namespace and entity name. Entities could be distinguished by comparing their QNames. In other words, in respect to the schema specification two entities from one and the same kind (e.g. type definitions) are considered equal if their target namespaces and names are equal.

With SAP Enterprise Services there are situations when a given entity – in most cases a type definition – is defined in one way within a Service Interface and the same entity is defined in a different manner (structurally) in another Service Interface. According to the schema specification they should equal to each other, since their QNames are one and the same. In reality the structural definitions of the entities differ,
which usually leads to unpredictable behavior at runtime during service consumption execution. The situation in SAP NetWeaver CE looks like that:

- SAP NetWeaver BPM and Service Composer have a solution to avoid type conflicts. This can be seen in the different namespaces after import.
- Web Dynpro does not provide a type reuse feature, so there cannot be any type conflict.
- JEE (proxy creation) provides a way to manually define the packages during export to avoid package conflicts during Enterprise Service proxy generation.
- CAF gets a problem, if two Enterprise Services that have conflicting namespaces are imported inside the same CAF project.

There are two services with the following XSD schema within its types section:

Type definition 1:
```xml
<xsd:schema ... targetNamespace=http://com.abc/types>
  <xsd:complexType name="MyType">
    <xsd:sequence>
      <xsd:element name="field_1" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Type definition 2:
```xml
<xsd:schema ... targetNamespace=http://com.abc/types>
  <xsd:complexType name="MyType">
    <xsd:sequence>
      <xsd:element name="field_1" type="xsd:string"/>
      <xsd:element name="field_2" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Here the type definitions, named MyType are conflicting ones (QNames equal to each other, internal structures – elements – are different) and if both services are consumed within one DC, there might be some consumption or classloading-related issues at runtime.

---

**Regarding Data Persistence (CAF, JPA)**

**Addressed issue:** the NetWeaver Composition Environment contains various programming models for persisting data. This section explains when to use which.

In SAP NetWeaver CE, the Composite Application Framework (CAF) is based on standard Java EE 5 features and consequently uses Java Persistence API (JPA) for object-relational mapping (ORM) and data storage.

JPA provides an object-relational mapping functionality that allows Java developers to concentrate on Java objects without needing to worry about how to store them in the database etc. All the low-level details dealing with JDBC drivers and (Open) SQL are shielded by JPA. The objects that are persisted are pure Java beans (POJOs).

The Composite Application Framework provides a design time modeling environment as a distinct Eclipse perspective within the NetWeaver Developer Studio (NWDS). Here, you can graphically model your Business Objects (BOs). During generation, the necessary database tables (in the Dictionary DC) and the corresponding Java classes (within the EJB Module DC) are generated automatically. This frees the developer from repetitive and tedious programming of CRUD operations and the like. In a nutshell, CAF generates a Stateless Session Bean, which acts as a façade and provides the CRUD operations.
Model-driven development tools like CAF are bound to define a common set of assumptions and standard use cases, which they support best, as otherwise the tools would become just as complex to handle as the underlying technologies.

The following is a description of the differences between the usage of CAF and pure JPA, separated in several categories.

**Join Clauses as Part of WHERE JPA-QL**

CAF does not support arbitrary (freestyle) WHERE clauses with SELECT, DELETE and UPDATE statements. The following restrictions are valid for CAF findBy operations. To explain the limitations a CAF Business Object Node with the attributes a and b is used.

1. Conditions could be defined over an attribute and permit the usage of the following operands: !=, ==, >, >=, <, <=, LIKE, BETWEEN.

2. The set of conditions defined over one attribute is disjunctive by nature, meaning that it consists of ‘OR-conditions’.

   **Example:** The variables p, q, r are different condition definitions over one attribute. They could then be bound together in a single WHERE clause ONLY in this way: WHERE (p OR q OR r)

   To make it more concrete, we would like to create a query that uses the attribute a for the parts in the query. Then the signature of the findBy methods looks like:

   \[
   \text{bo[]} = \text{findBy(a<operand, minValue, maxValue>[])}
   \]

   The parameter is an array for the different conditions for the attribute that are connected by an or-statement during query execution. Only those CAF Business Object Node instances that fulfill at least one condition are returned.

3. The set of conditions defined over two or more attributes is conjunctive by nature, meaning that it consists of ‘AND-conditions’.

   **Example:** The variable \(P_{x\_r}\) is a set of OR-conditions (see before) defined over attribute ‘x’. \(P_a\), \(Q_b\) and \(R_c\) could then be bound together in a single WHERE clause ONLY in this way: WHERE \((P_a \text{ AND } Q_b \text{ AND } R_c)\).

   To make it more concrete, we would like to create a query that uses the attributes a and b for the query, then the signature of the findBy method looks like:

   \[
   \text{bo[]} = \text{findBy(a<operand, minValue, maxValue>, b<operand, minValue, maxValue>)}
   \]

   The parameters are single conditions for the attributes that are connected by an and statement during query execution. Only those CAF Business Object Node instances that fulfill all conditions are returned.

4. From 1, 2 and 3, it is clear that CAF can only automatically build up the following type of WHERE clauses: WHERE \((p_1 \text{ OR } p_2 \text{ OR } \ldots \text{ OR } p_n) \text{ AND } (q_1 \text{ OR } q_2 \text{ OR } \ldots \text{ OR } q_m) \text{ AND } \ldots\)

   \[
P = \{p_1, p_2, \ldots, p_n\} - \text{all the conditions over a single CAF attribute}
   \]

   \[
Q = \{q_1, q_2, \ldots, q_k, \ldots, q_M\} - \text{all the conditions over another CAF attribute}
   \]

   To make it more concrete, we would like to create a query that uses the attributes a and b for the query with different conditions. The signature of the findBy method looks like:

   \[
   \text{bo[]} = \text{findBy(a<operand, minValue, maxValue>[], b<operand, minValue, maxValue>[])}
   \]
The parameters contain multiple conditions for attributes $a$ and $b$, connected by or statements during execution, i.e. only one has to be true to include the CAF Business Object Node instance into the result array.

5. CAF does not support Outer JOIN clauses.

**Domain Modeling / Database Design**

CAF also makes several assumptions and enforces certain rules during domain modeling. Those which could be perceived as stricter compared to JPA are listed below:

1. CAF does not support multiple attributes, i.e. lists. Multiplicity can only be achieved by using Business Object Node (i.e., JPA entity) associations.
2. If seen as database table relations, CAF associations are only $m \ldots 1$, or $m \ldots n$.
3. Single CAF associations ($1 \ldots m$ and above) are still persisted in a separate table. This might lead to poorer performance.

**JPA Fine Tuning and Configuration**

CAF supports limited configuration options of the JPA Persistence Manager.

1. ‘Lazy’ / ‘eager’ loading policy mechanisms cannot be controlled by CAF users. If the default value (lazy loading) needs to be changed, this can only be done by defining the value in JPA XML descriptors.
2. The JPA Entity Manager is capable of "caching" several database operations without directly performing these operations to the database immediately. In fact, the corresponding database operations (SQL statements) are performed once the EntityManager is flushed. This is normally done in alignment with the transactional context the operation runs in.

CAF flushes the cache after each operation. There is a partial improvement in NWDS 7.30 in terms of operations, which work on many instances of one and the same type at once, but the flush is still called after every operation. This behavior may cause performance issues, but it has been introduced in order to guarantee correct error handling if exceptions occur. A plain JPA application can choose when this use case would be most important and would flush the cache accordingly, probably not that often. As a consequence, CAF is not meant as a tool for mass database operations.

**NetWeaver CE 7.20 and below:**

```java
BONService bonService = ...;
Collection<BON> boNodes = ...;
for (BON boNode : boNodes)
{
    bonService.delete(boNode);
}
```

**After NetWeaver CE 7.30 and above:**

```java
BONService bonService = ...;
Collection<BON> boNodes = ...;
bonService.deleteMultiple(boNodes);
```
The following guidelines can be applied as a concrete recommendation for when using plain JPA is preferable to using CAF persistence.

1. If the usage of complex arbitrary queries would be necessary, choose JPA.
2. If flushing the Entity Manager cache after every operation is not desirable and/or DB modification performance is critical, choose JPA.
3. If the application database design would require use of relationships which are not supported by CAF, choose JPA.

Nevertheless, there are ways to bring CAF and JPA together, because CAF uses the Java EE standards. The interaction between the frameworks is based on the concept that JPA can reference the generated entities of CAF. For more details, see [9].

... Regarding Failure Handling

Addressed issue: failures can appear in all areas of a composite application. This section explains how to react in the different cases.

Failure handling can be required in both read-intensive and write-intensive scenarios, like reading from a second service if the original fails for some reason in read-intensive scenarios, or repeating a service call in write-intensive scenarios.

Based on the specific cause of the failure, we can separate three different flavors of failures and recommend different approaches for handling them:

- **Data validation failure** – This is when the (user) input data is not valid according to the system, and there is no way for this kind of failure to be automatically handled (fixed) by the system. It is expected that the user is able to fix the failure easily, for example by fixing the input data and trying again. Handling of the failure then means informing the user about the failure and maybe (if this is not already intuitive) providing hints on how to (manually) fix the problem. We recommend formalizing these types of failures through specific Java exceptions or Web service faults. Exceptions and faults could then be represented in a proper way by the UI part of the composite.

- **Process logic failure** – This is when there are certain known exceptions to the normal process flow, where a corresponding action could already be taken by the system according to the specific business process logic. The action itself could be an automated task (service call or trigger another process for example) or even a user (manual) task but the actual activation of the user task is triggered automatically based on the failure condition. We recommend modeling process exceptions via BPNN intermediate events/exception flows in SAP NW BPM if the business logic is already modeled as a business process. If the business logic is implemented in CAF or Java EE, we recommend handling exceptions to using the try-catch-finally clause accompanied by proper Java exception definitions. Service Composer (as in 7.20) does not offer this.

- **System failure** – This is when the normal (expected) work of the system is not possible due to a technical issue. An issue is technical when the end user cannot directly fix the main cause (meaning that it is not a matter of business data validity or business process conditions). This could be anything from misconfiguration of a system to hardware failure. It is not expected that the user can fix the problem directly, and it is likely that only users with special system authorizations (system administrator or account administrator, for example) or system support providers are able to fix it. We recommend informing the user (via a proper UI) about the problem and the immediate steps needed to fix the issue (configure system, call support center, file a bug report, etc.).

... Regarding Logging

Addressed issue: there are various ways of logging in a composite application. This section explains how to perform logging.

A composite application often implements one or more business processes. A commonly raised requirement is to provide information about the process execution from the business perspective. SAP NetWeaver CE is an assembly of several components, where no common facility (out of the box) exists to perform business
activity logging. The BPM Business Log is intended to provide relevant information about the process execution to the business user (in some cases, it can indicate where technical issues have occurred). To drill down to technical aspects of the process execution, the trace facilities of the involved components need to be inspected.

**Using BPM Business logs (applicable for release SAP NetWeaver CE 7.20 and above)**

If the whole business process is driven by or executed as a SAP NetWeaver BPM model, the business log can be leveraged to provide the requested information about the process. Using lean analytics, it is possible to provide better visualization of the business data in different reporting styles. Visual Composer can be used to provide a reporting UI for logged business data. The BI tool kit available with VC can use locally persisted business data as a data source. BI Tool kit uses BICS, which abstracts data sources from consuming clients.

**Using Locally Persisted Business Log (CAF BO)**

If BPM is not used, this type of business data logging functionality can be built using CAF BO. CAF and BI integration enables extraction of CAF services data and loading it into a BI system. The extraction of CAF applications data is provided by CAF runtime automatically. If you want to make custom data source and extraction logic, you should use application service BI extractor methods. The CAF runtime automatically provides a data source for each method and calls it when extraction is initiated. With this integration, it is possible to generate required reporting supported by SAP BI. The VC BI tool kit can also be used to generate reports.

**BICS - BI Consumer Services:**

- Abstracts clients from physical source (relational and multidimensional SAP and non-SAP sources: SAP BI, JDBC, XMLA)
- Provides analytical and planning capabilities (hierarchies, drill down, exceptions and writeback for example)

**Levels of Integration:**
• **Tight coupling**: This is a typical scenario with composition, where the BI application is embedded seamlessly in a composite application.

  - One integrated design time for transactional and analytical content
  - One single UI runtime that covers both transactional and analytical content
  - Seamless UI embedding
  - One backend session
  - Support for analytical and planning functionality
  - Examples: Visual Composer, Web Dynpro for Java, etc.

• **Loose Coupling**

  - Jumping from transactional content to standalone BI applications
  - Jump from tightly embedded applications to standalone BI applications
  - Target is launched in an extra window or in-place
  - Unidirectional eventing only
  - Examples of standalone BI applications that can be loosely coupled are:
    - Formatted Reporting (Crystal Reports)
    - Excel-based Analytics (BEx Analyzer)
    - Multidimensional Analysis (Pioneer)
    - Dashboards (Crystal Xcelsius)

**SOA Patterns**

**Addressed issue**: development in SOA requires specific application design patterns. This section describes the most common ones

Service Oriented Architecture is based on services and the assembly of services. There are therefore numerous ways to structure the services in the process area and the user interface area. The following concepts provide guidelines about how to structure the assembly and how the services are connected. The principles are structured in the usual way of architecture patterns in computer science [7], focused on SOA.

**ID Cross Referencing**

**Problem description**

Composite applications work with data from different backend systems. The data is retrieved or persisted using services (in this case it does not matter if Web service technology or RFC is used). But holding data in different backend systems and bringing it together by a composite application means that the origin of the data is not really changed. This means that data about a customer is still stored in one backend system, but data in other systems refer to it.

The composite application therefore brings together and tries to correlate data from various systems. Unfortunately, there is always the problem that a data set in one system is related with the data in another
A very common problem is that the different identifiers for the data sets have to be mapped between the systems.

**Pattern Description**

To solve the problem of ID mapping in a composite application, we recommend that there is a CAF BO that holds the mapping information, see [13]. The data structure is then always the same:

- Unique identifier in the composite application – key field
- Identifier of data in system A
- Identifier of data in system B
- …

If the composite application design is following the decoupling principles and has introduced a service contract and service contract implementation layer, see [Concrete Implementation of Loose Coupling in Service Consumption], then the mapping functionality of IDs is part of the service contract implementation layer, because all functionality that has to deal with different remote systems has to be part of the service contract layer implementation.

If the composite application is not implemented in the loosely coupled way, then the data in different remote systems has to be accessed from the composite application itself. Here the ID referencing problem is similar and therefore the following section explains how the pattern would be implemented in the tight coupling case. Nevertheless, the recommendation is to decouple a composite application as much as possible from the backend systems to be as flexible as possible (SOA principle).

If the composite application has to create new data in one system that is also transferred or used in another system, an entry in the mapping CAF BO has to be created. Usually, entering and requesting a new object in a remote system is done from a user interface component, meaning that the mapping CAF BO has to provide the access operations (at least the CRUD operations) in a way that can be consumed by the user interface layer. It does not mean that the handling of the CAF BO for ID storage is done in the UI, the persistency of the IDs and their mapping is an own entity in the composite application.

At a later point in time, when the process wants to work with the data, the mapping information should be retrieved, allowing the data retrieved from system A to be correlated with the data in system B (and vice versa).

Usually the ID referencing pattern is used in the service contract implementation layer (see loose coupling), because it is responsible for the abstraction from a concrete remote system, but it is not mandatory to couple both architecture patterns.

The pattern would look like this:

The pattern provides the following characteristics:
- Synchronously call service for local persistency
  - Fast
  - End user immediately gets feedback
  - Internal ID for request generation
- Process correlates the different IDs for the same data in different systems

**Example**

In real process modeling, the initial creation of the IDs looks like this:

![Diagram of process modeling](image)

The human activity is calling the CAF BO directly, so the relation to the CAF BO is not visible in the process modeling.

The parallel service calls with subsequent correlation ID persistency looks like this:

![Diagram of parallel service calls](image)

**Asynchronous Write**

**Problem Description**

This pattern generally describes how to establish remote write operations on services asynchronously. This decouples the frequently time consuming write operation from the process flow at the consumer side. Asynchronous operations and communication may be useful when method calls are invoked across process boundaries via remote mechanisms. In particular, when calling Web services using HTTP or similar remote protocols, network latency and bandwidth restrictions create communication bottlenecks. Asynchronous communication therefore will not hinder the process flow, but will just initiate the remote operation and processing of data.

**Pattern Description**

There are various flavours of the pattern.

1. The simplest case is the fire-and-forget sort of operation.
2. On the other hand, the consumer might want to get confirmation of the termination status of the operation either through active polling or notification. In order to phase the call and the possible downstream reply, an asynchronous operation needs to be supplied with a message ID. In case of a notification, the ID is part of the notification. For polling, the ID has to be part of the poll request. The
consumer always has to keep track of the ID, for instance through a call handle, an in-memory map or the like.

a. The polling version of the asynchronous write is depicted in the first figure below.
b. The notification version is shown in the second figure below.

Poll of completion status.

Notification of completion status.
Delayed Write

Problem Description
Composite applications are created based on a variety of technologies. All technologies have the potential to use services. This means that all layers can make calls to any backend, either synchronously or asynchronously.

A problem now arises if a service write operation is called directly from the user interface. This kind of direct call creates certain drawbacks in a composite application.

- There is no possibility to add additional steps, like approvals or checks by other users. If the write operation succeeds, and updating the process status fails, the process is in an inconsistent state.
- The write operations are usually more time consuming than the read operations because of locking in the backend systems etc. It is therefore advisable to decouple the time consuming operations from the UI to avoid waiting times for the user of the application.

Pattern Description
The user interface and the call of the write operation are decoupled. This means that the user interface is only responsible for getting the user input and for storing it in the process context. The write operation is performed as a subsequent step, meaning that the write operation is delayed in comparison to the solution if the user interface makes the call directly. This would look like this:

Decoupling the specific service calls does not come for free. Separating service calls into dedicated process steps causes additional effort, for error handling for example. Only modifying service calls should be separated into a subsequent process step, because these service calls change the backend data state. Further separation is not required, check services can be called by the UI component directly for example, because a check service is stateless and therefore does not impact the data consistency of the backend system.

Error Resolution: Compensation

Problem Description
Composite applications are the applications in the SOA landscape that have the task of calling services in different backend systems. A requirement that arises immediately is how to deal with operations that have to happen consistently in the different backend systems. A collection of operations like this is usually called a transaction, and all characteristics of a transaction are even expected in a SOA landscape [5].

To handle transactions is even not easy in one system. SAP NetWeaver CE provides support for consistent handling of resources by a transaction manager, such as database and JMS messaging. Unfortunately, this becomes nearly impossible in distributed and decoupled landscapes as we are facing it in SOA environments. One of the SOA principles is to decouple the business logic as much as possible. Bundling services that are not really meant to work together (because they are implemented and defined as different services). This is a conflict with the basic principles of SOA, because bundling of services is violating the decoupling principle.
The general problem is therefore how to ensure the consistency of different systems.

**Pattern Description**

The solution for achieving consistency in a SOA landscape with subsequent service calls is to introduce a compensation service [6]. A compensation service is designed to roll back all changes (even partial changes) made by a service call in the corresponding system. This means that the service and the compensation service always form a pair.

In real process modeling, this would mean that error handling always calls the compensation service if the original service call fails. So far, this is not a very complex infrastructure, as it only involves manual modeling for the process designer.

However, it becomes a little bit more difficult if there are subsequent service calls that belong together to achieve a consistent state in the connected systems. If one of the calls fails, all changes have to be rolled back. This would mean that service call A that happened before service call B and failed has to be rolled back too. In this case, the compensation services for B and A therefore have to be invoked, as otherwise the state in system A is not consistent with the state in system B.

Modeling for a pattern of this kind looks like this:

```
Service Call to System A
Compensate System A
Compensate System B

Service Call to System B
Compensate System B
...
```

**Error Resolution: Retry**

**Problem Description**

Since composite applications are applications that call services of different backend systems in a probably loosely coupled way, a requirement is that service calls may need to be retried in case the initial call fails. Usually in SOA, HTTP is used as the transport protocol. Within this protocol, it can happen that at the delivery of the information packages get lost. The retry of a service is accompanied by a retry policy. If the condition of the policy evaluates true, a retry is triggered, otherwise other measures like compensation patterns may take effect.

There are two kinds of failures for which the service consumer cannot be sure that the request has been processed properly.

1. Undelivered request
2. Undelivered response

Pattern Description
This pattern describes the infrastructure rather than a particular behavior of one of the involved components. Hence it is not a matter of how customers using services can resolve the issue, but it is rather the communicating and mediating components that are affected by this solution.

In general, the infrastructure setup can be done in one of the following ways:

1. **The service call is routed over an ESB (or comparable messaging mediator) to the service provider or an alternative provider.**
   a. The consumers and service providers communicate via an ESB which mediates between them.
   b. The ESB acts as intermediate service requestor.
   c. The ESB intercepts the request messages, evaluates the provider’s retry policy and may even determine alternative providers.

   Reliable messaging, as specified for instance by the WS-RM standard, is an example of such a sort of mediated communication.

2. **The service is idempotent and another call with the same payload and ID will not cause additional state changes in the corresponding backend system.**
   a. A service call / message is identified by a unique ID (e.g. UUID) attached to the message.
b. There are two fault scenarios:
   - The service provider receives the message and caches/persists the ID and sends the response back. However, the consumer will not obtain the message due to transmission errors.
   - The service provider does not receive the message due to transmission errors.

c. If the consumer does not receive the response, it retries the request with the same message ID.
   - The consumer evaluates the retry policy and retries its response accordingly.
   - The service provider provides the response message for the given ID. The payload for the response is either evaluated from scratch or retrieved from cached data in case the provider successfully received the request in a previous call (optional).

3. The service is not idempotent but will provide an operation by which the service consumer may poll or request again the response for the initially sent request.
   a. This pattern is very similar to the one for idempotent calls. The only difference is that a service provider has to keep track of already obtained messages / IDs and cache the responses.
   b. This pattern is similar to the previous pattern, but instead of sending the request again a light-weight repetition request is sent.
   c. The light-weight request contains the previous message ID. The provider either sends the response or a notification that the request for the respective message ID has not been arrived at the service provider. In the latter case the consumer needs to send another request containing the initial payload.

Like in the previous pattern, additional infrastructure both on provider and consumer side is required.

Modeled Alternative

Alternatively, the evaluation of conditions for retry policies can be handled in the process model itself. No particular requirements towards the infrastructure need to be taken into account. This evaluation precedes the service consumption. If the policy condition evaluates true and the service is then consumed successfully, the normal flow is followed. Otherwise, a compensation flow could be triggered, for instance. An example of this pattern is depicted below where a timer determines the timeout for a backend service call:
There are more variants of the pattern. For instance it might be crucial that both consumer and provider need to commit their state changes. In this case some sort of 2PC patterns may be applied in addition.

**Generic Data Transfer Object**

**Problem Description**

Composite applications are usually built on the paradigm that data is consumed via services. On top of these imported services, additional functionality like user interfaces, processes or additional business logic is executed. The data is accessed via imported services, possibly enriched through additional business logic, and then transferred up to the user interface or process layer.

If the used services are changed (e.g. an additional field was added), then the change has to be propagated through all the layers of the composite application. So, one small change possibly requires a long list of changes in the design of a composite application. The effort to do these changes increases the development time of composite applications.

**Pattern Description**

Usually the data transfer between the different layers of a composite application is done in a typed way. The data objects contain attributes and the data can be accessed in a type safe way.

But there is a possibility to define a generic data transfer object (DTO) without a typed structure. The DTO only contains key/value pairs, so that every layer can access it and can get the data.

Advantages of such a DTO:

- It can contain any amount of data
- Additional data following a new structure of the data object can be simply added to the key/value pair storage
- Every layer can access the data

Disadvantages of such a DTO:

- Access works in an untyped way, therefore access is not type-safe and access errors fail only at runtime

Such a DTO can be implemented in SAP NetWeaver CE in the following ways:

- **EJB**: a POJO with key/value pair data type can be defined as access structure for the methods of the EJB.
- **CAF**: in CAF the data structures are defined within the framework. After that the Java structures are created based on this definition. If there is a data structure for key/value pairs, it can be used for a business object and corresponding access methods or as data types for application services.
Related Content


[8] Matthias Steiner: Building Extensible Composite Applications with SAP


