Do's and Don'ts with SAP BPM

Applies to:

SAP NetWeaver 7.4, EHP 1 for SAP NetWeaver 7.3, SAP NetWeaver 7.3; SAP NetWeaver Composition Environment 7.2; EHP1 for SAP NetWeaver Composition Environment 7.1;

Summary

This document shortly outlines the best practices for implementing SAP NetWeaver Business Process Management (SAP NetWeaver BPM). In this document, you will also find Do's and Don'ts for different components and learn how to best handle business requirements to avoid ineffective practices.

Many of these recommendations are explained in details in the documentation mentioned at the end of the document as a reference.

Company: SAP AG

Created : October 2011

Updated: June 2014

Table of Contents

Composition Environment – Architecture

Business Data / Leading Artifact

Guideline:

Carefully choose the location where to store business data – in the process context, the user interface (UI) session, an SAP NetWeaver CE local business entity (Composite Application Framework (CAF)/JPA), or remotely in the backend. If strict data consistency is not required, avoid remote persistence.

Motivation:

Each type of data storage implies a certain life cycle for the data. Each option also has its performance implications. Properly choosing the leading artifact for the business problem at hand helps steering the composite project in the right direction.

Software Component / Development Component Granularity

Guideline*:*

- Use Software Components (SCs) to structure your deployment;
- Create only as many SCs as you really need to cover your distribution requirements;
- If the entire composite is to be executed on one server, use exactly one SC
- Use Development Components (DCs) to structure your development;
- Define DCs along functional criteria, not organizational factors;
- Carefully note that DCs are built as a whole, not incrementally, so having more DCs is better;
- Don't be too fine-granular because each DC has a build overhead;
- Put components with identical dependencies in one DC;
- Group components accessed by only one other in one DC.

Loose Coupling

Guideline*:*

- Prefer loose coupling (clean interfacing, asynchronism, protocol agnosticism, ...) to tight coupling;
- Use an intermediate service layer to access enterprise services;
- Place the service layer into a separate SC;
- Only resort to tight coupling if it is considered safe, i.e. flexibility is not needed, or if performance is an issue.

Motivation:

Loose coupling gives you flexibility in deployment and agility in reacting to changes (at the expense of a higher complexity). Thinking about the degree of decoupling in advance avoid unwanted surprises later on. The intermediate service layer can be used for service mocking and simplification. The latter reduces the amount of secondary data.

Extensibility

Guideline*:*

- Use extensibility to allow modification-free (ex)change of parts of your application;
- Use the extensibility framework for EJB, CAF, and Web Dynpro (WD);
- Mark a form as extensible in the Adobe Interactive Forms (AIF) case;
- Invoke sub-processes via their Web service interface in case of SAP NetWeaver BPM.

Motivation:

Being able to customize certain predefined parts of an application without having to modify it can be highly beneficial, specifically in organizations with an ISV role.

External Libraries

Guideline:

- Make external libraries accessible via a dedicated external library DC;
- Expose the libraries as both assembly and compilation public parts;
- Refer to the latter from all places where you need to compile against the libraries and use the former for assembly into an enterprise archive. If you assemble the library DC into a standalone enterprise archive, let other archives establish a run time dependency onto it.

Motivation: Embedding external libraries via the SAP component model is a complex task.

ID Cross-Referencing

Guideline:

- Use CAF BOs to persistently correlate data originating in different remote systems;
- Combine CAF Bos with an intermediate service layer to abstract from backends.

Motivation:

Composite applications work with data from different backend systems. Very often, these data need to be correlated to each other.

Asynchronous Write

Guideline:

- Decouple backend communication using asynchronous service calls;
- In case a confirmation is needed, employ either polling or notification (push) patterns.

Motivation:

Remote backend calls, specifically if following the document-style SOA paradigm, take a considerable time to execute and create communication bottlenecks.

Delayed Write

Guideline:

- In case of a UI embedded into a process, limit service calls to read services without side effects;
- Delay other service calls to a separate automated activity.

Motivation:

Calling write services or services with side effects directly from a UI makes it hard to properly handle errors in the process. This hides the business logic from the process (transparency issue) and lacks the flexibility a process has to deal with error situations, e.g. suspend – fix issue (e.g. edit process context) – resume. Also, locking mechanics employed in write services can lead to long response times in user interfaces.

Error Handling – Compensation

Guideline*:*

Use compensation activities to ensure overall consistency in distributed landscapes and roll back (partial) changes made by previous erroneous service call(s).

Motivation:

Distributed scenarios often suffer from the absense of transactions across system boundaries. Using compensation activities allows ensuring eventual consistency in case of errors.

Error Handling – Retry

Guideline*:*

- In case of idempotent services, use retry policies enforcable on a WS-RM-compliant WS RT or messaging mediators such as PI / ESBs to handle connectivity issues during service invocations or, implement the retry policy at the process level;
- With non-idempotent services, try offering a second service on the service provider side to check the status of the service invocation instead. This pattern can be combined nicely with compensations.

Motivation:

In case of idempotent services, retry policies can be very handy to compensate unsuccessful service invocations that may arise due to connectivity issues (e.g. package loss in case of HTTP protocol).

SAP Composition Environment – Installation and Update

Update from SAP EHP1 for SAP NetWeaver CE 7.1 to SAP NetWeaver CE 7.2

Most common mistakes:

DON'T Restart from scratch an update that has started updating the system (unless you're planning to restore the system).

Motivation: There's no such thing as an update which cannot continue

DON'T Update SAP JVM upfront (or if you do make sure you don't miss anything).

Motivation: The update tools will handle updates of the JVM. There is no need to do that manually. There is a risk to do a lot of mistakes if you perform the JVM update manually.

DON'T Update with JSPM.

Motivation: JSPM is not supported for SAP NetWeaver CE updates.

DON'T Update JSPM upfront.

Motivation: There's no such thing as an update which cannot continue

DON'T Use *stack.xml*.

Motivation: UMS makes no use of it.

DON'T Copy the executables directory of one system to another. If you do, watch for the *uninstall.properties* file.

Motivation: There is system-specific information in the exe directory, it's not just binaries.

Recommendations:

Guideline:

Always make sure that the system has enough disk space before upgrade.

Motivation: The free space in database should be around 50GB approximately for a successful upgrade.

Guideline:

Make sure you have the right set of components for your system (Inbox updates) and specially make sure you don't miss the Kernel and SAP JVM.

Motivation:

If you miss the Kernel and SAP JVM (specially) the update will fail. If more components are in the inbox they will simply be deployed to the system.

Guideline:

Check for component versions on the old releases after the update is complete.

Motivation: It might happen that a component is missing from the inbox directory. If the component is missing it will not be updated.

Upgrade to Composition Environment as part of SAP NetWeaver 7.3

Recommendations

Guideline*:*

Always make sure that the system has enough disk space before upgrade.

Motivation: The free space in database should be around 50GB approximately for a successful upgrade.

Guideline*:*

DB upgrade should be done.

Motivation: While upgrading from lower releases like SAP EHP1 for SAP NetWeaver CE 7.1 and SAP NetWeaver CE 7.20 to SAP NetWeaver CE 7.30, the database should also be upgraded beforehand after checking the PAM document, else the upgrade may fail stating this reason

Guideline*:*

Create another user with administrative rights in windows.

Motivation: SAP installer does not install with the logged in user as ADMINISTRATOR hence we need to create another user (Alias for user administrator) and proceed

Guideline*:*

Make sure you select the option to update the SAP JVM to latest version in Maintenance Optimizer (MOPZ).

Motivation: SAP EHP1 for SAP NetWeaver CE 7.1 and SAP NetWeaver CE 7.20 come with lower version of SAP JVM but SAP NetWeaver CE 7.30 requires SAP JVM 6.1 and above, which needs to be selected while downloading the required files using MOPZ.

For more details about MOPZ go to https://service.sap.com/mopz.

SAP BPM – Modeling Best Practices

Complexity, Performance, and Sizing

***Guideline**:*

- Keep process model complexity at 25-50 process steps;
- Model block-oriented wherever possible;
- Name your primary data[1] as such;
- Minimize the amount of secondary data[2] in your process;
- Design new interfaces with small signatures;
- Keep the number of attributes to be mapped at any one interface below 50.
- Use 'move-corresponding' wherever possible;
- Re-use human tasks wherever appropriate;
- Consider data mappings as a severe runtime performance factor;
- Close unused IDE projects.

Motivation:

With a highly comfortable tooling environment such as NWDS, you can easily be tempted to disregard the implications of your actions. For example, if mapping an overly complex data structure can be done with one gesture, you won't reason whether you really need all the data in it and what runtime impact it might have.

Task Handling

***Guideline**:*

- Reduce the number of tasks by generifying similar tasks into one.
- Use expressions within a task to customize the task subject/description based on the process context.

Motivation: Tasks have a complex life cycle and they are technically implemented as processes. Reducing the number of tasks can help improve overall performance (DT & RT).

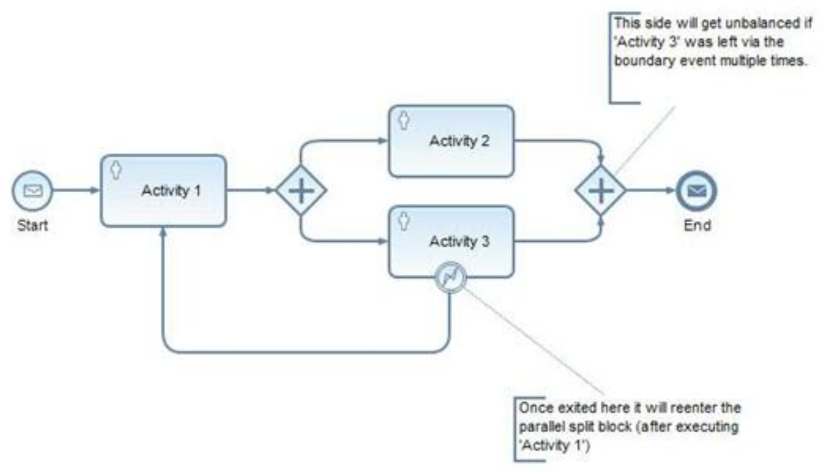
[1] Primary data – This is the data that drives the process and is used for example in decision gateways, as input for service parameters of the automated activities, etc.

[2] Secondary data – The secondary data is related to the primary data but it is not directly involved in the process execution and could be stored in a place different from the process context.

Gateways

***Guideline**:*

Avoid mixing gateways


This side will get unbalanced if 'Activity 3' was left via the boundary event multiple times.
Start
Activity 1
Activity 2
Activity 3
End
Once exited here it will reenter the parallel split block (after executing 'Activity 1')

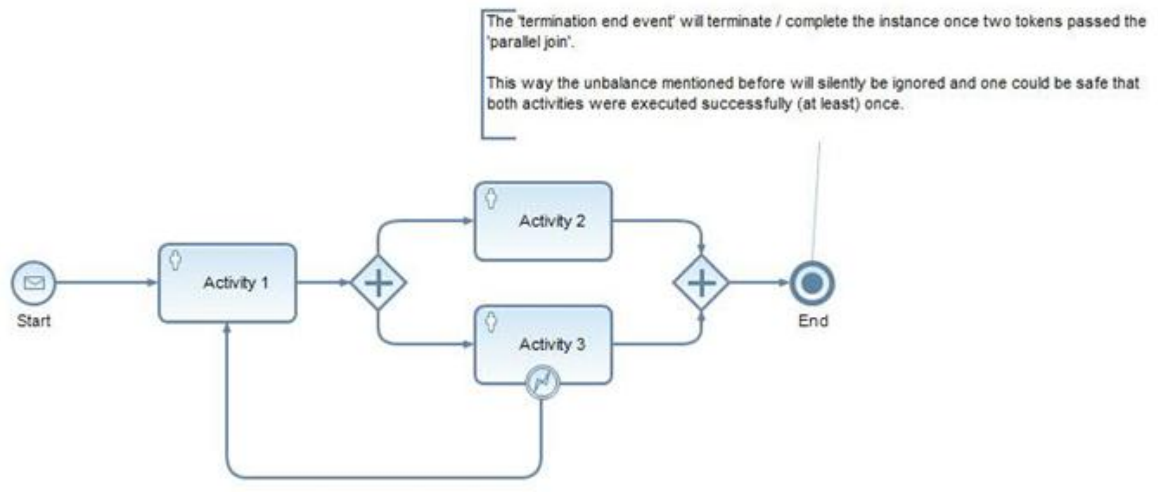
Motivation:

The "split" gateway will cause parallel execution of "Activity 2" and "Activity 3". Activity 2 passes and the token will wait before the "join" gateway. If Activity 3 fails and leaves via the boundary event the complete "split / join" block will be reentered again. Now both activities pass (order does not really matter). As a result there will be 2 token on the one incoming "join" branch and 1 token on the other incoming branch. The "join" will merge one token from each branch into one outgoing token and the end event will be reached.

The 2nd token on the upper branch will still be there and will never be consumed, thus the process will never be "completed".

Possible solution:

Use Termination End Event. The process instance will be terminated once the outgoing token reaches the end event. Note that this is only applicable if it is required that both activities are executed (at least) once.


The 'termination end event' will terminate / complete the instance once two tokens passed the 'parallel join'.
This way the unbalance mentioned before will silently be ignored and one could be safe that both activities were executed successfully (at least) once.
Start
Activity 1
Activity 2
Activity 3
End

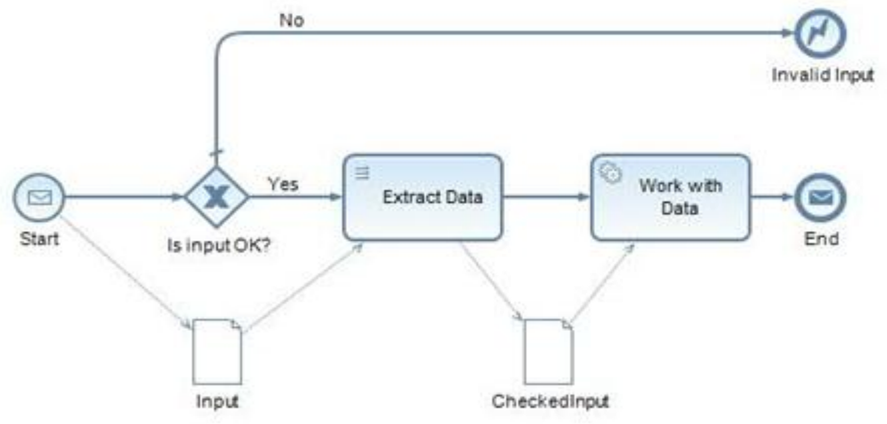
XSDs with Constraints

***Guideline**:*

Do not rely on XSD constraints (e.g. "string10" a custom string that only allows 10 characters) in custom types. If possible check the input explicitly and also expose error conditions in order to handle them (e.g. a human step that allows correcting the data if applicable).

Motivation:

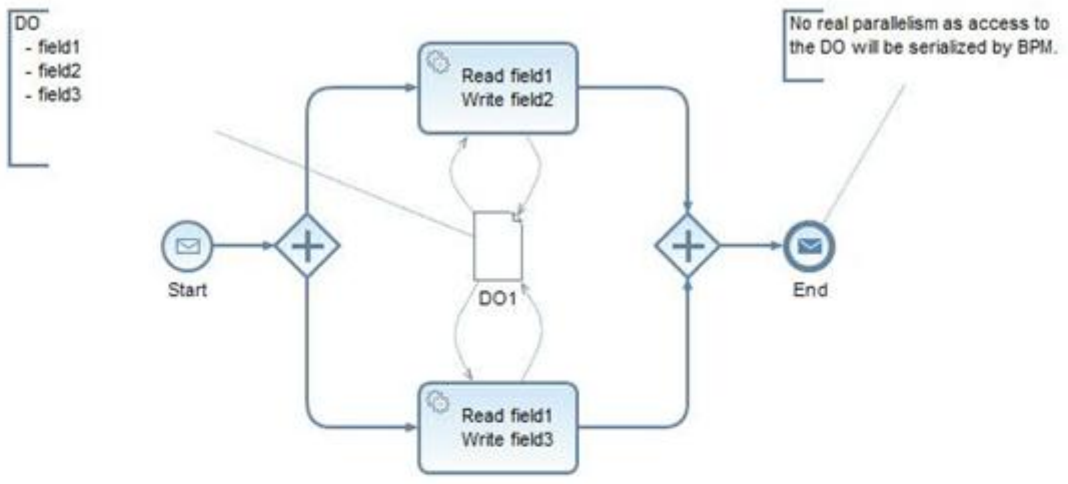
If XSD constraints will throw exceptions at runtime which one cannot recover from (e.g. an 11 character string will never fit into your type).



Data Object

***Guideline**:*

Avoid Parallel Access On a Data Object.

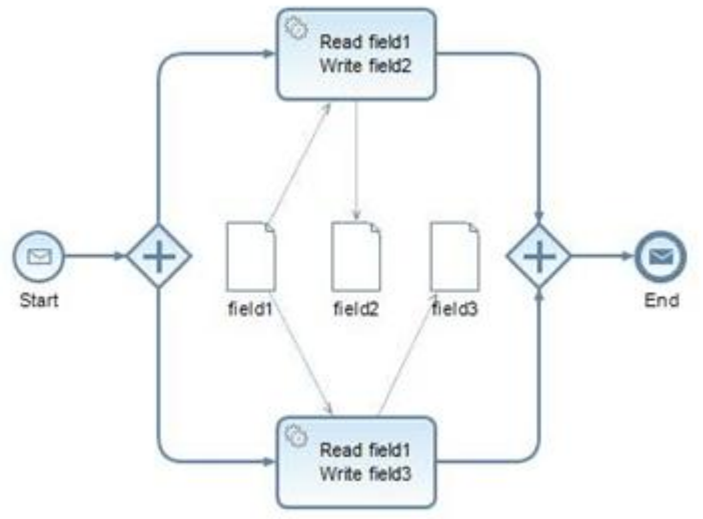


Motivation:

Both parallel activities access Data Object "DO1" that consists of 3 fields. The engine will serialize access to the Data Object. The desired parallelization will not take place.

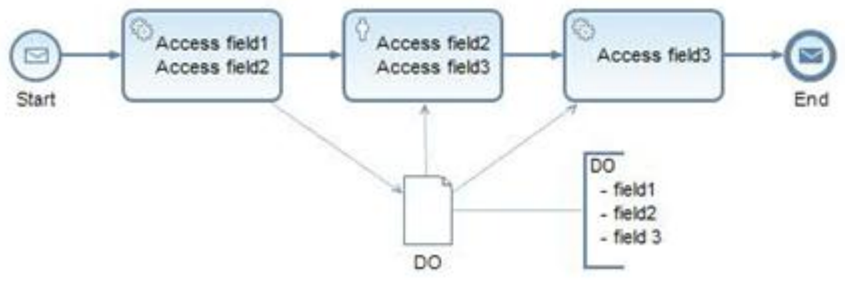
Possible solution:

Work with small data objects that only contain the relevant parts (trade-off). This way both activities can read from the same Data Object in parallel. Write access (to different parts of the former Data Object) takes place in different Data Objects and can happen in parallel.



***Guideline**:*

Avoid large Data Objects.



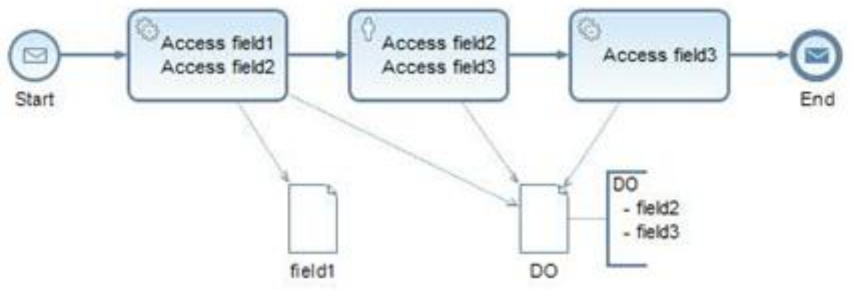
Motivation:

Again all (sequential) activities work on the same Data Objects. This time it's not about parallelism, but asynchronous processing. The Data Objects contains parts that are not needed after the first step.

If such a Data Objects would be reloaded (e.g. after an output mapping from the human activity) all unnecessary data will be retrieved as well as it is part of the structure.

Possible solution:

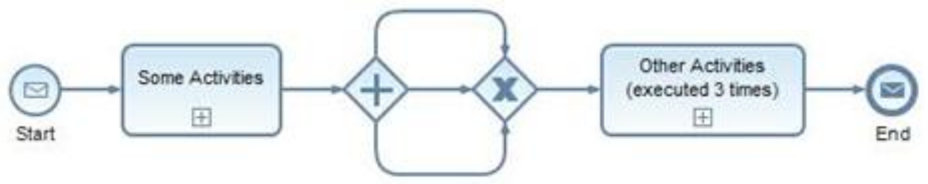
Splitting it up and only keep the parts that are needed for the asynchronous call does the trick. This way only data that is relevant after the asynchronous execution will be loaded into memory again.



Thread split

***Guideline**:*

Don't use "static" tread split.



Motivation:

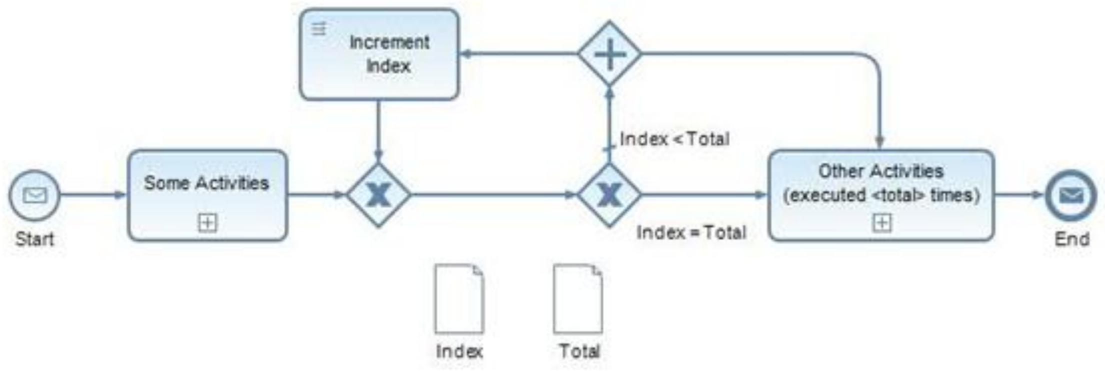
The "split" will generate a token for each outgoing connector. The "merge" will let any token pass (means all 3). This initiates a referenced sub-process 3 times (in parallel). The first sub-process that completes (returns a response) will trigger the message end event.

Possible solution:

Explicitly model 3 activities that do the same and perform a „join" before the „message end event".

***Guideline**:*

Don't use "dynamic" tread split.



Motivation:

Number of instances can be set at runtime (input mapping into "total"). As long as "index < total" it will increment the index by 1 and start off a thread in parallel. Once the last iteration is reached the "choice" will kick off the last thread.

The first thread to return a response will trigger the message end event. Have a look at "thread merge".

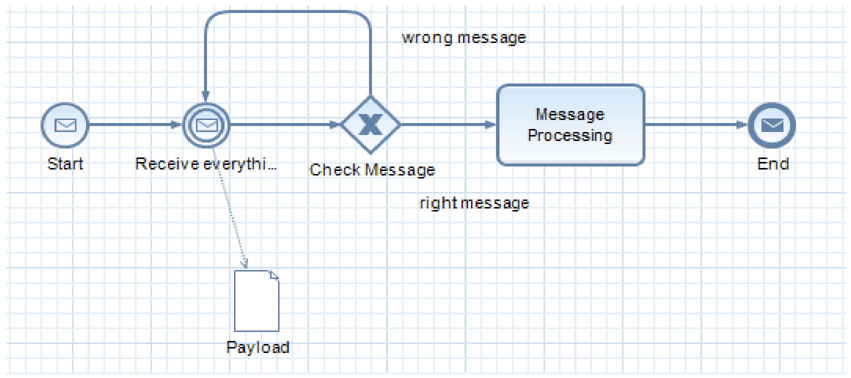
Possible solution:

Use „Parallel For Each" loop option of the SAP NetWeaver BPM instead. When you use „Parallel For Each" loop make sure to check in advance how many items the collection for the loop has.

Correlation

Guideline:

Don‘t do message filtering inside processes.

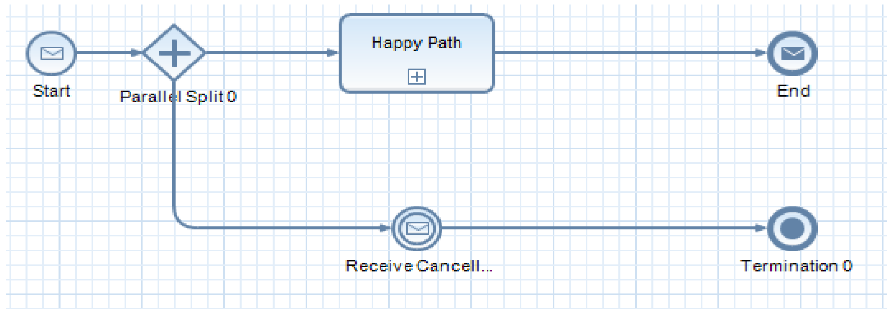


Alternative:

Make correlation conditions are selective (specific) as possible

Guideline:

Don‘t abuse IMEs to re-implement process control functionality that already exists in NWA (e.g., cancel running process instance)

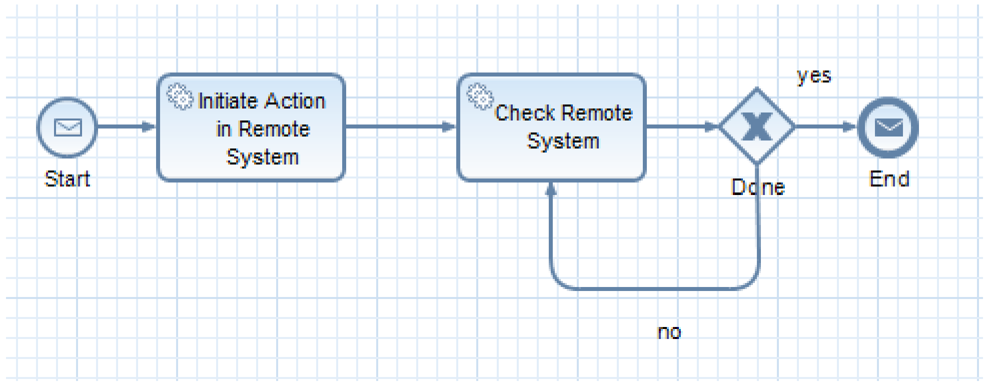


Alternative:

Use (1) the BPM API or (if not present in there) (2) use the existing runtime tools (NetWeaver Administrator).

Guideline:

Callback from conversation partner (remote application or system) - No abuse, but still costly



Alternative:

Sometimes, it may be less costly to rather poll the conversation partner if it has completed the action that was triggered before (instead of being called back).

Performance Tuning

Guideline:

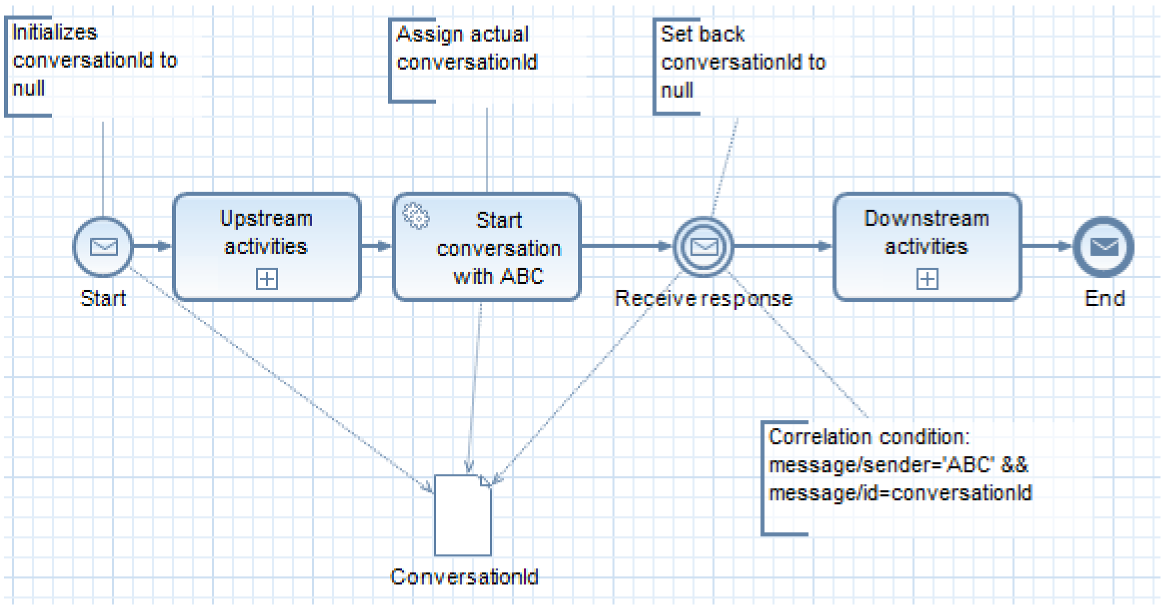
Avoid broadcasts to multiple processes by re-using message triggers in multiple processes

Alternative:

Rather use different message triggers (if semantically suitable)

Guideline:

Keep matching scopes only as short as strictly needed


Initializes conversationId to null
Assign actual conversationId
Set back conversationId to null
Start
Upstream activities
Start conversation with ABC
Receive response
Downstream activities
End
Correlation condition: message/sender='ABC' && message/id=conversationId
ConversationId

Parallelism

Guideline:

Try to minimize shared Data Object access in concurrent branches

Motivation:

To avoid locking and to improve performance

Guideline:

Use the "Parallel For Each" loop if you have to process multiple equally structured items in a similar way or want to process a large number of data items in a concurrent fashion. For instance you have to start a survey process for each employee in your department or you have to process a large number of sales orders.

When you use „Parallel For Each" loop make sure to check in advance how many items the collection for the loop has.

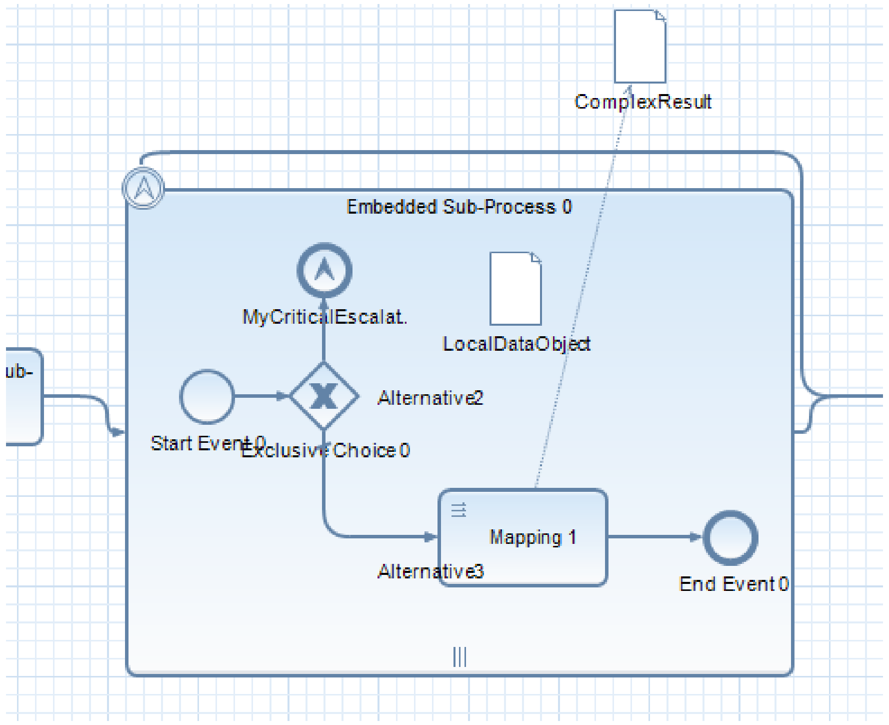
Motivation:

At creation time there is a synchronous creation of tokens and line data objects for each instance. After that the processing gets asynchronous and is controlled by the Galaxy Kernel, means the concurrency is determined by the available working threads and possible lock collisions.

Higher memory footprint, increased locking collision probability and processing time

Guideline:

Try to map content back to a process data object inside a loop. This way the access to that data object is again serialized by the general locking mechanics.

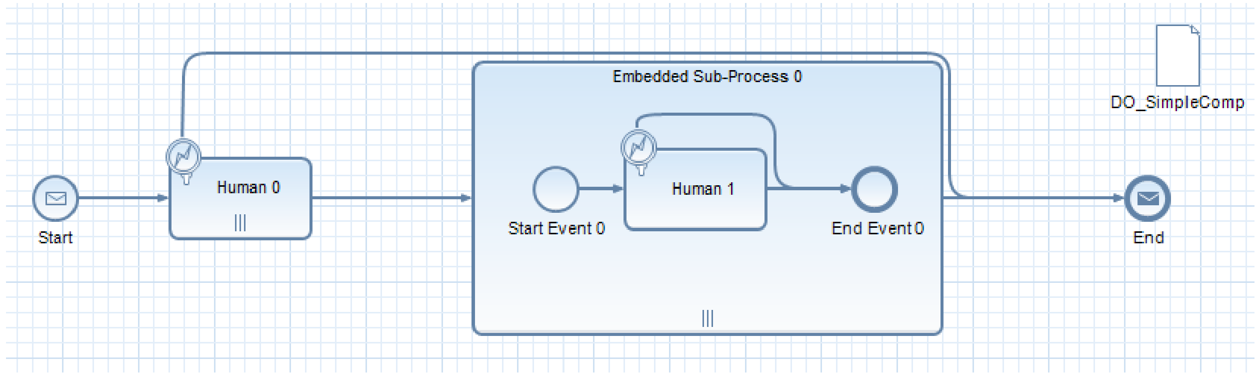


Motivation:

Line items are copies of the data, so fewer locking collisions on the payload as with modeled concurrency that operates on one data object with the use of mapping functions and indexes.

Guideline

If boundary events are needed on activity level, surround the activity with an embedded scope and use multi instance on the ESF.

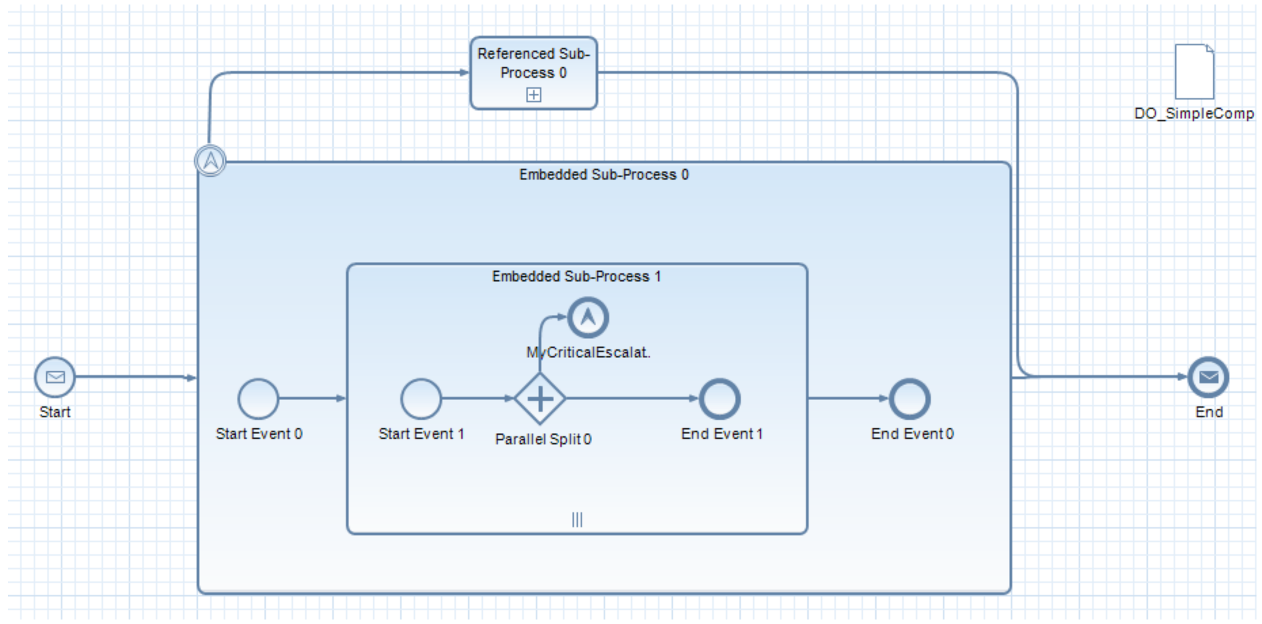


Motivation:

Critical boundary events are defined on loop level, so critical exceptions would cancel all loop cycles.

Guideline

Avoid modeling of error end events inside multi instance activities that trigger nearly at the same time.



SAP BPM – User Interface Design and Development

UI Flow

***Guideline**:*

- Use WD UI flows in conjunction with SAP NetWeaver BPM processes to express consecutive steps performed by a single user that represent a single consistent task from a process p.o.v. (free of side effects and potential errors except for user input validation);
- Resort to standalone WD UIs in case of self-initiated activities;
- Use a dedicated WD UI calling a process‘ WS endpoint to instantiate the process with a user-initiated action.

Motivation:

Using WD UI flows can be combined nicely with processes to keep users from revisiting their worklist and avoid process cluttering caused by consecutive tasks.

UI Modeling

***Guideline**:*

- Use Visual Composer (VC) to quickly build simple, purely model-driven UIs or UIs built on top of SAP BI data sources (also for SAP NetWeaver BPM analytics);
- Use WD Java for more complex tasks: extensibility, programmed UI logic, stateless or asynchronous UIs;
- Use WD to leverage Adobe Interactive Forms;
- Embed WD UIs into VC models through black box integration if needed.

Motivation:

VC and WD are UI technologies with different focus areas. Knowing when to use what avoids stepping into the pitfall of having chosen the wrong technology (no migration).

UI Lifecycle

***Guideline**:*

- Process Context data will be available inside the UI component context after wdDoInit() when the WD component is loaded;
- Prevent code after Complete Event signalization by the component;
- Do not forget to create at least one start point and one end point (for VC components) / completion event (for WD)

UI Versioning

***Guideline**:*

UI components do not have a runtime versioning support. Any changes to UI components must be compatible with deployed process definition (interfaces, events). In case of incompatible change, reimport the UI into process model (e.g. by using the refresh button in the task editor), and mappings adapted accordingly

Data Transformation

***Guideline**:*

- Use CAF, SC and EJB to achieve loose coupling;
- Use SC to adapt services involving simple to moderately complex mappings and CAF otherwise;
- Use CAF specially if local persistency or custom coding is needed in conjunction with adaptation;
- Avoid using SC for implementing business logic as error handling is not supported yet (resort to CAF or SAP NetWeaver BPM in that case);
- Don‘t use SC for high-performance scenarios;

- Exposing EJB as a web service is the recommented way to implement loose coupling for the use cases that require the best performance
- Notice CAF's inability to handle QName uniqueness type conflicts.

Motivation:

CAF and SC are model-driven tools with different runtime models and characteristics. Depending on the use case one or the other might be more appropriate.

Data Persistence

***Guideline**:*

- Use CAF for standard persistence scenarios where highest possible performance is not needed and out-of-the-box conventions established by CAF are suitable, i.e. relations are stored in a separate table, finder methods follow the *(p_1 OR p_2) and (q_1 and q_2)* style, equi-joins are used, and flushes take place immediately;
- Use JPA specifically if performance needs to be tweaked, standard table layout is not acceptable, or complex queries are required. In that case, use the JPA Editor to model JPA entities and apply templates to generate CRUD session beans.

Motivation:

CAF aims at supporting the standard use cases of data persistence without requiring detailed JPA knowledge. Knowing the boundaries of CAF before deciding to use it is crucial to avoid getting trapped in them later on.

Failure Handling

***Guideline**:*

- Handle data validation errors at the UI level and grant users the possibility to retry if possible;
- Make sure to provide clear Java exceptions or WS faults to denote such errors if service calls are involved;
- Handle expectable business logic errors via BPMN using intermediate/boundary events in case of SAP NetWeaver BPM processes;
- Use proper exceptions together with CAF/EJB in programmed cases;
- In case of unexpected system failures, inform and instruct the user properly.

Motivation:

Failure handling can be required in both read- and write-intensive scenarios. Depending on the nature of the failure, different remedies are advised.

Logging

***Guideline**:*

- Leverage the SAP NetWeaver BPM business log together with reporting activities and the VC BI kit / WD to perform logging and analysis locally on the SAP NetWeaver CE server at the SAP NetWeaver BPM level (tightly coupled);
- The BI tool kit available with VC can use locally persisted business data as a data source. Use CAF BOs to store logging data and (custom) extractors to extract it from a SAP NetWeaver CE system into a BI system. Then, use the VC BI kit / Web Dynpro as before, with the difference the data sources reside on the BI system (loosely coupled).

Motivation:

Composite applications employ different logging mechanics across the different layers. Moreover, depending on the use case, loose or tight coupling of analytics might be the preferred option.

SAP BPM – User Interface integration

Data Context

***Guideline**:*

- Decide which data to store in process context, and which data to retrieve when User Interface (UI) is initialized , recommendation is to keep the process context as small as possible
- Expose only the minimum set of data what is needed by BPM in the UI interface definition
- Ensure up-front validation of data that the user enters on the UI by plausability checks (format, range, etc.) in order to avoid process errors

Supported Data Types

***Guideline**:*

- Use the Built-In Simple Types.
- Unsupported Data Types are further simple WD Types, CCTS or Java Native Types as listed in the documentation and OSS note 1450393

Facet support (e.g. Enumerations, Value Constraints, Patterns, etc.)

***Guideline**:*

Avoid generating UIs for interfaces with Facet support and also avoid usage of UIs that neglect the same

Motivation:

Facets are not supported for task UIs.

WD Model Nodes

***Guideline**:*

WD Model Nodes can be used if they are not exposed via the Local Component Interface Controller Context

Date and Time

***Guideline**:*

SAP NetWeaver BPM internally works on date and time information in UTC. UI components should expect and provide this and provide the display information according to the end user’s time zone offset.

UI components which are assigned to tasks are loaded and processed for in- and out-coming data dynamically.

Type compatibility needs to be provided between the following:

- XSD
- SDO
- Web Dynpro
- VC

Especially for the case where an application server is located in a different timzone than its users it is important to note that the data exchange for date information is based on UTC time. The display of date information is handled by the UI application and its specific technology taken the end users timezone into account.

Data transfer

***Guideline**:*

At Runtime SAP NetWeaver BPM processes rely on the data interfaces that were imported while modeling at design time. Dynamic data context modification in the UI component is not supported.

Unified Data Handling

Guideline:

WD-Java "Context Nodes have to be unique within scope of the entire context."

On-line and Offline Forms

Guideline:

Use one form for one task.

Motivation:

After the task completion the form turns invalid and cannot be used for tasks again (in other processes as well as in the same process instance).

Guideline:

Prevent to set up multiple BPM instances to re-use one mailbox.

Motivation:

Each BPM server instance requires its own dedicated mailbox for incoming offline forms (service user mailbox).

Related Content:

Installation and Upgrade Guide for SAP Enhancement Package 1 for SAP NetWeaver 7.3 and SAP NetWeaver 7.3

Installation and Upgrade Guide for SAP NetWeaver Composition Environment 7.1 EHP1 and 7.2

Composite Development Architecture Guidelines

Business Process Management on Help Portal

Business Rules Management on Help Portal

SAP NetWeaver BPM on SDN

SAP NetWeaver BRM Best Practices

SAP NetWeaver BPM Resource Center

Copyright