hybris software

An SAP Company

hybris software
An SAP Company

hybris Developer Training Part I - Core Platform

# Process and task engines

hybris software
An SAP Company

# Architectural Overview

## Features

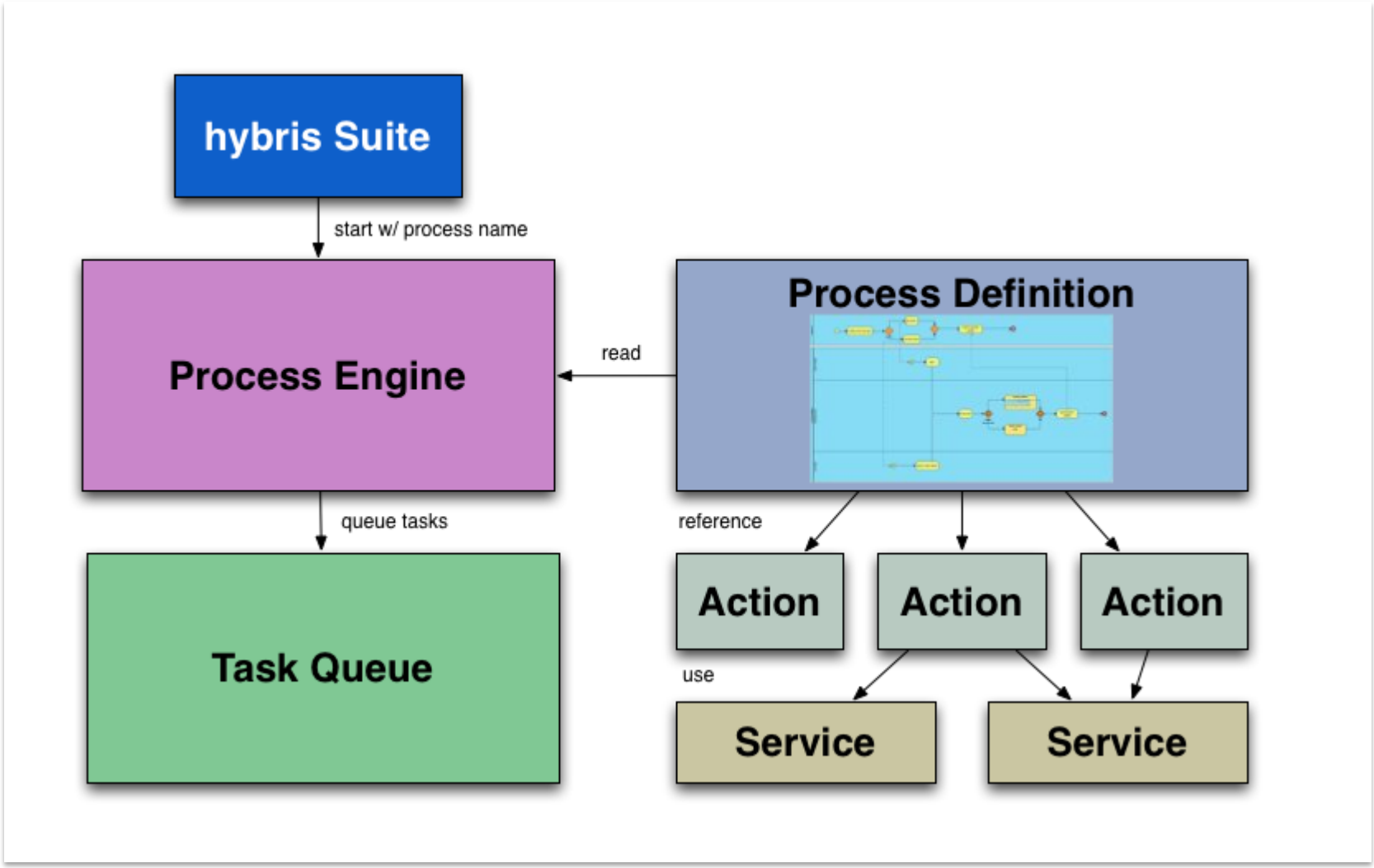
## Business analysis

## Creating a Process

## Scripting support

## Task Extension

# Architecture of the Process and Task Engines



hybris software
An SAP Company

Architectural Overview

# Features

Business analysis

Creating a Process

Scripting support

Task Extension

# Process Engine Features

- The hybris process engine extension allows you to model, support and monitor business processes
- The process engine interprets a process definition consisting of nodes and transitions
- A process is defined in an XML file and can be run asynchronously
- Actions are performed in a certain order and are only carried out if all predefined conditions are met
- You can wait for events, notify users or user groups, fire specific actions and determine subsequent actions based on action results

hybris software
An SAP Company

Architectural Overview

Features

# Business analysis

Creating a Process

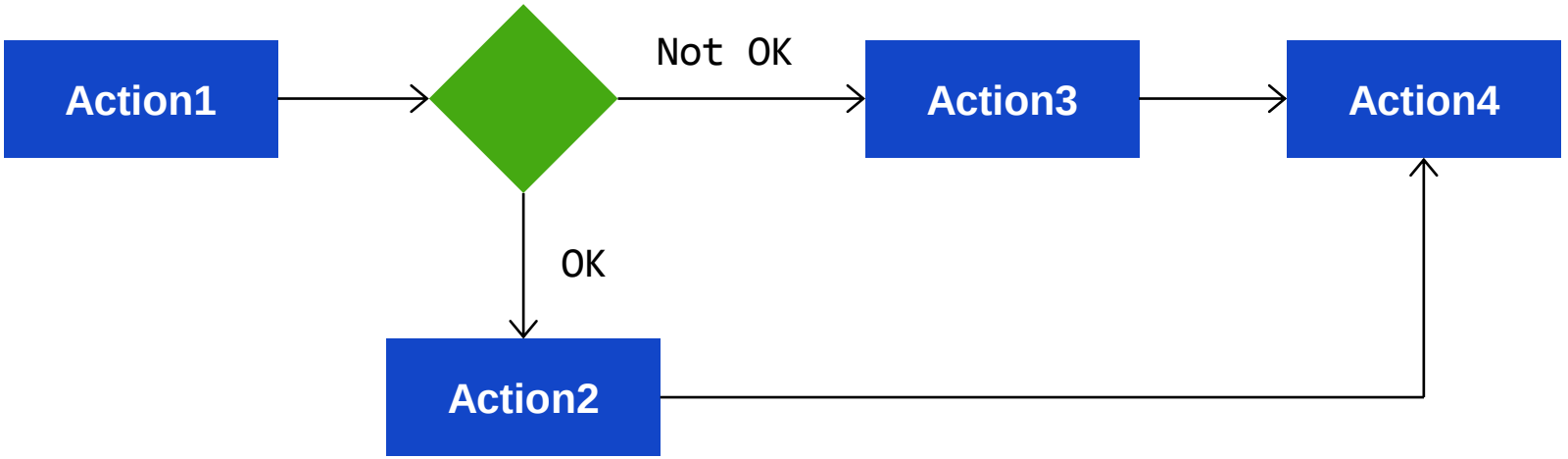
Scripting support

Task Extension

# Business analysis

- The first step in working with the process engine extension typically consists of conducting a business analysis

- A flowchart permits to define a step-by-step solution to a given problem:



hybris software
An SAP Company

Architectural Overview

Features

Business analysis

# Creating a Process

Scripting support

Task Extension

# Process definition file

The workflow from the analysis is translated into a process definition XML file:

```
<process name="Example" start="Action1">
   <action id="Action1" bean="Action1">
      <transition name="OK" to="Action2"/>
      <transition name="NOK" to="Action3"/>
   </action>
   <action id="Action2" bean="Action2">
      <transition name="OK" to="Action4"/>
   </action>
   <action id="Action3" bean="Action3">
      <transition name="OK" to="Action4"/>
   </action>
   <action id="Action4" bean="Action4">
      <transition name="OK" to="success"/>
   </action>
   <end id="success" state="SUCCEEDED">Everything OK</end>
</process>
```

# Node Types (1)

- ***Action nodes*** - carry out process logic and permit alternative actions to be carried out

```
<action id="isProcessCompleted" bean="subprocessCompleted">
   <transition name="OK" to="sendCompletedNotification"/>
   <transition name="NOK" to="waitForSubprocessEnd"/>
</action>
```

- ***Wait nodes*** - wait for a subprocess or an external process result

```
<wait id="waitForSubprocessEnd" then="isProcessCompleted"
         prependProcessCode="false">
   <event>${process.code}_SubprocessEnd</event>
</wait>
```

# Node Types (2)

- ***Notify nodes*** - inform a user or user group of the state of a process

```
<notify id="notifyAdminGroup" then="split">
    <usergroup name="admingroup" message="Do something!"/>
</notify>
```

- ***Split nodes*** - split the process into parallel paths

```
<split id="split">
    <targetNode name="doActionA"/>
    <targetNode name="doActionB"/>
</split>
```

- ***End nodes*** - end the process and store state in a process item

```
<end id="error" state="ERROR">All went wrong.</end>
<end id="success" state="SUCCEEDED">Everything is fine</end>
```

# Process Actions

- The next step consists in defining all actions specified in the bean attribute of individual action nodes
- Actions are the most important part of the process engine extension and implement logic or call specialized services
- An action performs a single piece of work and produces an action result which is directed as input to the next action
- All actions from the process definition XML file are implemented as actions classes

# AbstractSimpleDecisionAction

```
public class Action1 extends AbstractSimpleDecisionAction
{

    @Override
    public Transition executeAction(BusinessProcessModel process)
    {
        if(...)
            return Transition.NOK;
        else
            return Transition.OK;
    }
}
```

# AbstractProceduralAction

```
public class Action2 extends AbstractProceduralAction
{
    @Override
    public void executeAction(BusinessProcessModel process)
    {
        modelService.save(process);
    }
}
```

# Using a common context

- A common context can be used for all actions belonging to a particular process
- This is achieved with the class `BusinessProcessModel` or one of its subclasses (e.g. `ForgotPasswordProcessModel`)
- An instance of this class is passed as a parameter each time an action is called:

```
public void executeAction(final ForgotPasswordProcessModel process)
```

- The process fills this context with values
- You may add your own attributes to `BusinessProcessModel`

# Integration with Spring

Finally, we need to add the action classes to `<my-extension>-spring.xml` file:

```
<bean id="Action1" class="org.training.actions.Action1"
                                    parent="abstractAction"/>

<bean id="Action2"class="org.training.actions.Action2"
                                    parent="abstractAction"/>

<bean id="Action3"class="org.training.actions.Action3"
                                    parent="abstractAction"/>

<bean id="Action4"class="org.training.actions.Action4"
                                    parent="abstractAction"/>
```

# Creating and Starting a Process

- To create a new process instance, you need to call the `BusinessProcessService` method `createProcess()`.

```
businessProcessService.createProcess( processName )
```

- To start a process you need to call one of the available `startProcess()` methods of the same `BusinessProcessService`.

```
businessProcessService.startProcess( businessProcessModel )
```

# Dynamic process definition and versioning

- A process can also be defined at runtime and stored in the database.
- A **DynamicProcessDefinition** can be created in hmc/backoffice under Scripts -> Dynamic Process definition
- DynamicProcessDefinition contains an attribute **version** which is updated automatically when the process definition is changed.
- If you update the definition for an existing process, the previous version will be backed up automatically.
- The DynamicProcessDefinition also contains a boolean attribute **ActiveFlag** which is used to point to the active (most recent) version of the process definition.

hybris software
An SAP Company

Architectural Overview

Features

Business analysis

Creating a Process

**Scripting support**

Task Extension

# Scripting support for Business processes

**With the support for scripting, it is now possible to define, start and monitor business process completely at runtime.**

→ The business logic of an Action can be scripted directly within the process xml, declared as a `scriptAction`.

```
<?xml version='1.0' encoding='utf-8'?>
<process xmlns='http://www.hybris.de/xsd/processdefinition'   start='action0'
name='testProcessDefinition'>
    <scriptAction id='action0'>
        <script type='javascript'>
            (function() { return 'itworks' })()
       </script>
        <transition name='itworks' to='success'/>
    </scriptAction>
    <end id='success' state='SUCCEEDED'>Everything was fine</end>
</process>
```

# Scripting in Business process – contd.

- You can also access the process context in an `scriptAction`:

```
<scriptAction id='action0'>
<script type='javascript'>
var parameter = process.contextParameters.get(0);
parameter.setValue('changedFromScript');
modelService.save(parameter);
'itworks‘
</script>
<transition name='itworks' to='success' />
</scriptAction>
```

- The process can be started dynamically using the Scripting languages console in hac

```
businessProcessService.startProcess("testProcess1","testProcessDefinition")
```

hybris software
An SAP Company

Architectural Overview
Features
Business analysis
Creating a Process
Scripting support

# Task Extension

# Foundation used for single-process engine step

- The hybris Task extension:
  - allows you to schedule time- or event- driven tasks by placing them on a task queue
  - provides a lighter weight approach to tasks than CronJobs. But offers less functionality
  - processes tasks asynchronously
  - tasks execution is distributed across the cluster to ensure reliable tasks processing
- A task can also be defined and executed as a Script.

# The Task Engine

- The Task extension is designed to be used in a `ServiceLayer`-based application
- It provides a service for scheduling new actions and triggering events: `TaskService`
- Actions are defined as Spring beans in an extension's Spring configuration file e.g. `myextension-spring.xml`
- To define an action, you implement the `TaskRunner` interface
- In a clustered environment you may specify which nodes should process tasks:
  - `task.workers.max=10`

# Quiz questions

1. Give a brief architectural overview of the process engine extension.
2. What is typically the first step in defining a new process for the process engine extension?
3. How do you define a process?
4. Name three different types of nodes.
5. What is the most important field in a node?
6. How do you implement logic for the process engine extension?
7. Compare the hybris Task extension to a CronJob and highlight some of the differences.

# References

- wiki.hybris.com/display/release5/processengine+-+Technical+Guide
- wiki.hybris.com/display/release5/Order+Management+Tutorial+-+An+Example+Process
- wiki.hybris.com/display/release5/task+-+Technical+Guide