Integration of Activiti BPM

Request For Specification (RFS)

Project: ***OpenWMS***
Customer: *OpenWMS*

Project Nr: *P0001*
Document Nr: *10b-04*

Version: *0.2*
Date: *14.05.11*
Status: *draft*

Table of Contents

1 Scope....2
2 Terms and Definitions....2
3 Rationale....2
3.1 Current Situation....2
3.2 Problems....3
4 Vision....3
5 Proposed solution with Activiti BPM....3
6 Next steps....4
7 Appendix: Revision History....5

Illustration Index

Illustration 1: Overview and sequence of actions....4

1 Scope

This document focuses on the TMS module for now.

2 Terms and Definitions

TMS

The Transport Management System (TMS) is a separate sub module of openwms.org and responsible for processing events from underlying PLC (programable logic controller) subsystems. These systems control the hardware and mechanics and coordinate sensors (e.g. light barriers) and actors (like motors). But higher level control logic is hosted on an upper layer, the TMS running on a server hardware. That means routing and dispatching of TransportOrders has to be done in TMS. The interaction between PLC subsystems and the TMS layer is performed with driver components (implemented in Java). Most PLC systems communicate over TCP/IP with the upperlying TMS, the same in openwms.org, but this can change some day to another type of protocol. The way how PLC and TMS communication is done in practice does not affect the topic adressed here and is not part of this document.

Events

In general PLC systems are sending event messages to the TMS (ACSII telegram strings over TCP/IP, transformed by a driver component to events). These events are captured in TMS and force the execution of actions. Due to TMS is implemented in Java we expect an event as a simple Java class. The event can be seen as a trigger of a flow execution in Activiti.

Locations

Beside events a Location is part of the trigger, too. In particular a Location is the physical place within the system where an event occurs. That means that an event is always coupled to the Location where it occurs and is part of the same PLC telegram message.

Actions

Basically the event and the Location is enough to choose the appropriate action. Within this scope, the action should be a sequence of simple Java methods of arbitrary classes.

3 Rationale

3.1 Current Situation

So far the implemented action is a kind of procedural sequence of methods. In openwms.org we use the Spring Framework to manage all Java classes as beans, hence the action is a hard-coded invocation of bean methods with some if-then-else logic in between.

Example:

```
transportUnit.move(currentLocation)
TransportOrder to = transportOrderService.createTransportOrder(transportUnit)
Location targetLocation = null;
if (to.hasAsNextTarget()) {
    sendResponse(to.getAsNextTarget());
} else {
    targetLocation = locationService.getDefaultLocation(currentLocation);
} ...
```

3.2 Problems

Code like above is very specific to each project and has to be rewritten again and again. There is no reusability and the stable part (the service methods) is mixed into the unstable changing code. The composition of business service methods is mostly defined in a workflow within a high-level specification document and has to be implemented in Java language. This approach is error-prone, less efficient, hard to survey and not reuseable.

Due to this reason we need to separate stable and approved service methods from the dynamic workflow and try to put the dynamic part into an own domain specific language that allows a better definition of the workflow.

4 Vision

- Differentiate the often changing dynamic code from the static and proven service methods.
- Services and methods should not know about nor depend on their runtime environment and the workflow engine. Beans must not extend any framework specific superclasses nor implement such interfaces.
- It must be easy to offer service methods as workflow actions. Probably through an own Java annotation.
- Implementing the workflow is done within a higher language, in graphical or textual style (BPMN?).
- Changing the workflow can be done in real-time without the need of compilation or redeployment the application.

5 Proposed solution with Activiti BPM

I already did some research on the opensource BPM project Activiti (www.activiti.org) and it seems to be a proper candidate to solve the problems described in this document. Activiti integrates very well with Spring and ist very flexible and customizable.

I ran some of the examples that come with the Activiti download and read the documentation about integration and extensibility. It is possible to create Service Tasks of own Java classes, but these classes have to extend an Activiti superclass (JavaDelegate). Another solution would be to access own Spring managed beans via expression language. But this looks cumbersome as well.

I would see a solution with an own Spring BeanPostProcessor (BPP) that analyzes all of our Spring beans and recognizes an own class-level annotation (like @Workflow). The BPP collects all relevant data of such beans (like public methods with arguments and return values) and stores it in Activiti's database.

Probably it becomes necessary to introduce a thin adapter layer between Activit and our Spring beans.

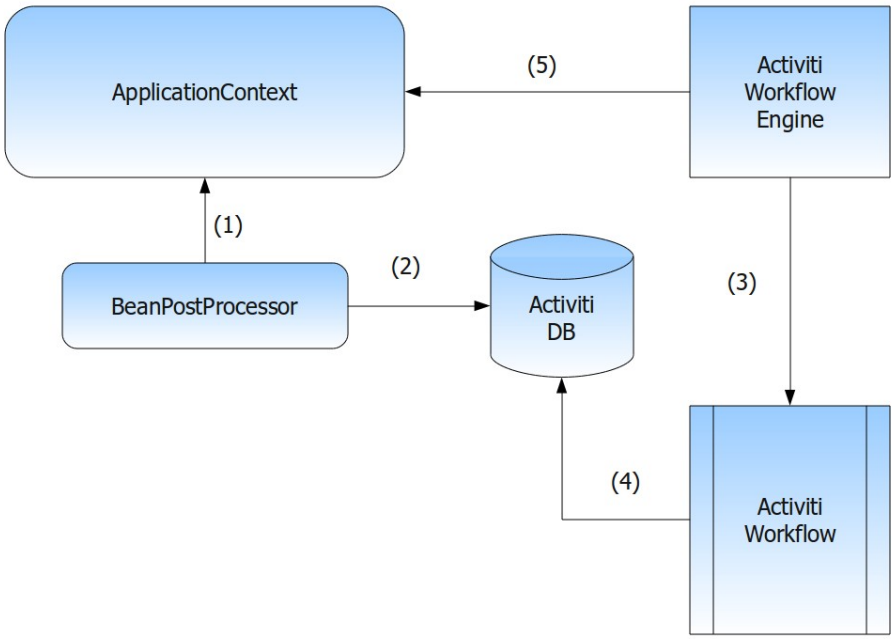

Illustration 1: Overview and sequence of actions

Step	*Description*
(1)	When the ApplicationContext is started an own BeanPostProcessor post processes all Beans and looks for an own annotation @Workflow
(2)	The information about @Workflow beans and their public methods is written into the Activiti database
(3)	When a workflow is executed, the Activiti engine reads the workflow definition
(4)	The workflow definition (in BPMN) consists of Java Tasks. These tasks are methods of our @Workflow beans
(5)	The Activiti Engine processes the workflow and calls the @Workflow beans from the started ApplicationContext

6 Next steps

Setup Activiti and play around with the delivered examples. Focus on the Spring integration and find out how the resolution of Java classes works in detail and where Activiti stores this information internally. Have a loook at the extensibility of this mechanism. Make a design proposal for a proper solution that fits our need. Have in mind that our application is split into several application contexts which means we need to have something like a service registry available for all bundles (application contexts).

7 Appendix: Revision History

Version	*Date*	*Author(s)*	*Description*
0.1	04/13/11	scherrer	initial version
0.2	05/14/11	scherrer	Added diagram and sequence description