Integration of Re-configuration/Re-scheduling Algorithms into Process Architecture

Deliverable D5.2

FFG – IKT der Zukunft
SHAPE Project
2014 – 845638

Table 1 Document Information

Project acronym:	SHAPE
Project full title:	Safety-critical Human- & dAta-centric Process management in Engineering projects
Work package:	5
Document number:	5.2
Document title:	Integration of Re-configuration/Re-scheduling Algorithms into Process Architecture
Version:	0.1
Delivery date:	E03/2016 (M3)
Actual publication date:	01.04.2016
Dissemination level:	Public
Nature:	Technical Report
Editor(s) / lead beneficiary:	Siemens
Author(s):	S. Sperl
Reviewer(s):	G. Havur, A. Polleres

Table 2 History

Version	Changes	Authors
0.1	Created Structure	S. Sperl

Contents

1 Introduction

This document is part of work package 5 (WP5) on integration of adaptivity algorithms into the process architecture of the SHAPE project. Two main scenarios have been developed and prototypically implemented: Re-configuration/Re-scheduling (cf. Section 2) and Document Generation (cf. Section 3).

2 Adaptivity in Scheduling

2.1 Technical Details

Details of the integration of the resource allocation algorithm and the translation from BPMN to Answer Set Programs (ASP) can be found in the diploma thesis ″Formalisms and Tools to Describe and Monitor Engineering Processes″ in the following chapters:

- Chapter 3: Resource Assignment and Allocation under Constraints in BPM
- Chapter 5: Integrating Resource Allocation Capabilities into a BPMS

2.2 Integration with Camunda

There are two components that are to be integrated with Camunda. First the Scheduler, then the component that during execution returns the scheduled user to a startable task.

The scheduler can potentially be started at any point of the process execution (it is a JaveServer Faces (JSF)[1] page), but is currently limited to process starts (because of GUI limitations). Below a startEvent is configured in such a way as to require the assignments (schedule) to be done before it can be started (if configured like this the process actually has to be started by the JSF page).

```
<bpmn2:startEvent id="StartEvent_1" camunda:formKey="app:assignments.jsf">
```

There are multiple ways to set the assigned user of a task once it becomes required.

- By using process variables (not recommended once it gets complex).

[1] http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

```
<userTask id="task" name="My Task"
          camunda:assignee="${processVarWithUser}"/>
```

- By invoking a bean

```
<userTask id="task" name="My Task" camunda:assignee=
          "${assignmentService.getScheduledUserofTask(execution)}"/>
```

- Or by setting it in a TaskDelegate during the create event of a UserTask.

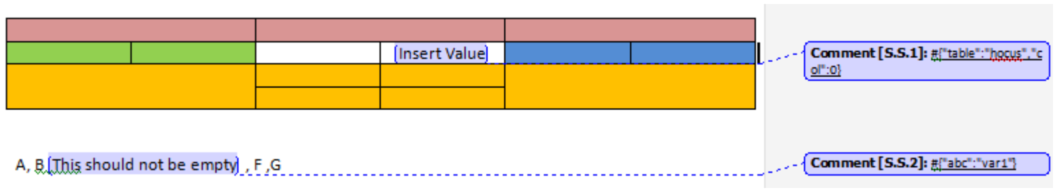
```
<userTask id="task1" name="My task" >
  <extensionElements>
    <camunda:taskListener event="create"
        class="org.camunda.bpm.MyAssignmentHandler" />
  </extensionElements>
</userTask>
```

In general invoking a bean is likely the best approach in terms of flexibility and ”effort in manually editing BPMN files”.

3 Adaptivity in Monitoring and Documentation

The railway industry requires extensive and reliable documentation as part of it’s validation process. Currently a lot of potentially trivial information has to be maintained and cross checked manually by experts. In order to improve this process we implemented an adaptive documentation system.

In it’s basic conception, parts of an existing document can marked as ”to be filled out”, the system will then adapt the document to include the currently available information. As marking mechanism the comment system was chosen, since this allows for the intuitive error reporting (problems are appended to the comment).

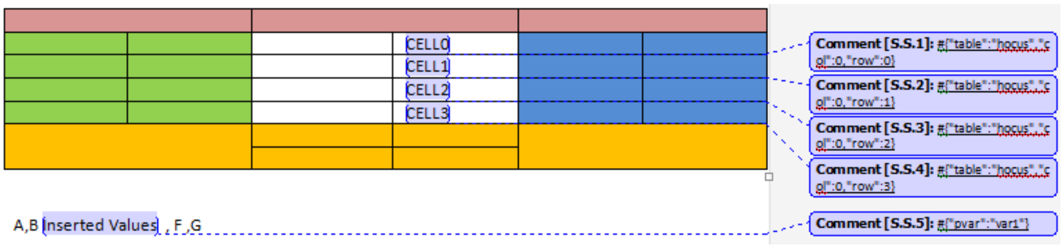

Insert Value

Comment [S.S.1]: #{"table":"hocus","col":0}

A, B This should not be empty , F ,G

Comment [S.S.2]: #{"abc":"var1"}

Figure 1 Exemplary MS Word Template, updateable regions are marked by word comments



CELL0
CELL1
CELL2
CELL3

A,B Inserted Values , F ,G

Comment [S.S.1]: #{"table":"hocus","col":0,"row":0}
Comment [S.S.2]: #{"table":"hocus","col":0,"row":1}
Comment [S.S.3]: #{"table":"hocus","col":0,"row":2}
Comment [S.S.4]: #{"table":"hocus","col":0,"row":3}
Comment [S.S.5]: #{"pvar":"var1"}

Figure 2 Exemplary MS Word Result

3.1 Technical Details

These operations are built upon the Apache POI project, which allows editing of MS Office documents via Java. Sadly the MS Word integration is likely the least feature complete of all office formats, for our purposes we extended the official release with our requirements and posted a patch [1].

The Interpreter expects the documents executable Comments to start with a # and then a valid JSON expression, all single underlined. If the JSON expression is invalid, an error will be reported, but comments not starting with # while single underlined will be ignored entirely.

The project is `at.shape.docgen`, primary components are `Function` and `Interpreter`.

```
public interface Function {
   public boolean executeable(XWPFRun r, SpinJsonNode n);
   public void execute(XWPFComment c, XWPFRun r, SpinJsonNode n);
   public String getName();
}
```

Via `execute()`, `Function` must inform the interpreter, if this `Function` can (should) work with the provided input data. The `XWPFRun` represents the commented text region and the `SpinJsonNode` represents the parsed JSON term that has to be in the comment.

In addition, `execute()` supplies the `XWPFComment` since implementations likely wish to overwrite the JSON expression contained in the comment (to stop re-interpretation or to store a machine readable version of the displayed data).

The `getName()` function is currently only used for error reporting purposes and should give a readable description of the problem.

In general it is advisable to design `Functions` in such a way:

```
assert(f.executable() == true);
assert(f.execute())
assert(f.executable() == false);
```

This helps with preventing unnecessary repeated executions (though you could also just reduce `maxPasses` to 1).

```
public class Interpreter {
   public void execute(XWPFDocument document);
   public ArrayList<Function> getFunctions();
   public int getMaxPasses();
   public void setMaxPasses(int maxPasses);
}
```

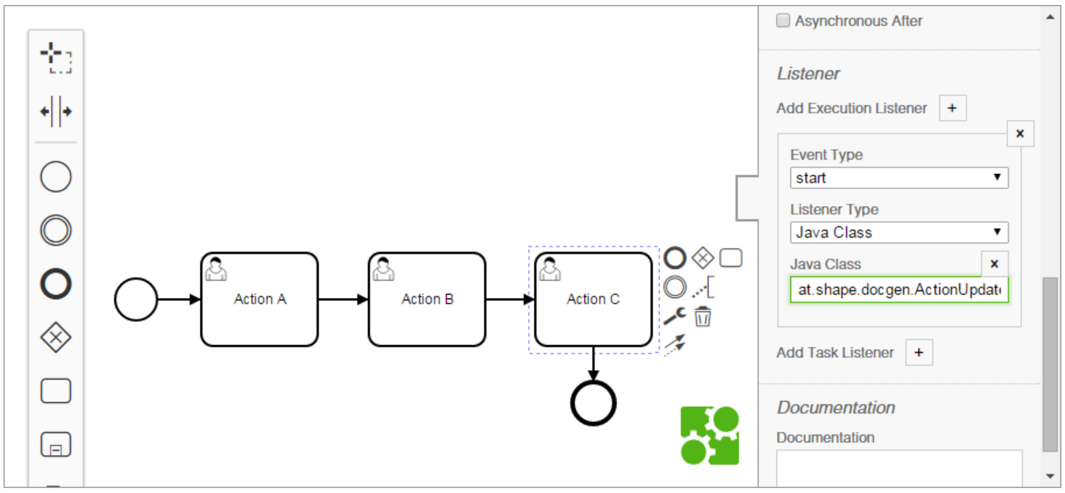
The `Interpreter` can be executed on an entire `XWPFDocument`, `getFunctions()` returns the list of `Functions` that the `Interpreter` will use on the document. If an executable Comment has no or more than one `Function` that match, an error is reported (and no further action is taken on this comment). The interpreter can also be configured to run multiple passes with the same set of functions (and will stop early once there are no more executable comments).

The library also ships with 3 pre-made `Functions` that allow simple replacement, growing a table by a number of rows, and convenient filling of grown table cells. Example:

```
Interpreter interpreter = new Interpreter();
interpreter.getFunctions().add(new
    NamedReference("variablemarker",keyValueMap));
interpreter.getFunctions().add(new TableExtender("tablemarker",10));
interpreter.getFunctions().add(new TableReplacer("tablemarker"){
   @Override
   public void execute(XWPFRun r, SpinJsonNode row, SpinJsonNode col) {
      r.setText("CELL" + (col.numberValue().intValue() +
          row.numberValue().intValue()),0);
   }
});
interpreter.execute(document);
```

3.2 Integration with Camunda

In general, execution listeners can work at any place in the program. Cf. Fig. 3.



Asynchronous After

Listener

Add Execution Listener +

Event Type: start

Listener Type: Java Class

Java Class: at.shape.docgen.ActionUpdat

Add Task Listener +

Documentation

Documentation

Figure 3 BPMN side update handler for document generation

References

1 Simon Sperl. Apache POI Patch. https://bz.apache.org/bugzilla/show_bug.cgi?id=56469, 2016. [Online; accessed 17-March-2016].

2 Alexander Wurl. Formalisms and tools to describe and monitor engineering processes. Master's thesis, Vienna University of Technology, January 2016.