D6.1.2 Smart Process Execution (Prototype II)

Authors: UVI (lead)

Delivery Date:	Due Date:	Dissemination Level:
2013-10-07	2013-08-31	PP

This deliverable describes the second prototype implementation of task 6.1. As stated in the Description of Work (DOW), this deliverable is a prototype (software) deliverable. As such, this document is reduced in length and its only purpose is to briefly describe the prototype functionality as well as to give installation instructions and usage clarifications. This document will be shipped together with the software itself.

SEVENTH FRAMEWORK PROGRAMME

Document History	
Draft Version	V0.1, UVI, July 26th, 2013 V0.2, UVI, September 30th, 2013 V0.3, UVI, October 2nd, 2013 V0.8, UVI, October 4th, 2013 V1.0, UVI, October 7th, 2013
Contributions	**UVI** - Jürgen Mangler - Tobias Hildebrand
Internal Review 1	TIE, Juanvi Vicente Vidagany Espert, TIE, October 3rd, 2013
Internal Review 2	Dieter Schuller, TUDA, October 4th, 2013
Final Version	October 7th, 2013

Table of Contents

List of Figures

Executive Summary

The Smart Process Execution (SPE) component will be at the heart of ADVENTURE, as it will orchestrate all interaction in a Virtual Factory. Its purpose is to execute Smart Processes, modelled in the Process Designer (D5.1). The second prototype (Prototype 6.1.2) is expanding the first prototype (D 6.1.1), in several dimensions. The enhancements of the second prototype cover the areas developer friendliness, the support for ADVENTURE specific events, a script task extension, an instantiation extension, an ADVENTURE specific extension that allows the translation of BPMN 2.0 models into the internal execution language and an adaptation extension. These enhancements are described in more detail in Section 1.2.

This deliverable provides a description of the second prototype implementation of task 6.1. The prototype includes all the necessary sub-components described in the technical specification (D3.3). As this document describes a prototype (software) deliverable, it is reduced in length and just briefly describes the functionality as well as highlights installation instructions and usage clarifications. This document will be shipped together with the software itself.

1 Introduction

ADVENTURE – ADaptive Virtual ENterprise manufacTURing Environment – is a project funded in the Seventh Framework Programme by the European Commission. ADVENTURE creates a framework that enhances the collaboration between suppliers, manufacturers and customers for industrial products and services.

1.1 ADVENTURE Project Aims

The framework proposed by ADVENTURE provides mechanisms and tools that facilitate the creation and operation of manufacturing processes in a modular way. ADVENTURE combines the power of individual factories to achieve complex manufacturing processes. It provides tools for partner-finding, process creation, process optimization, information exchange as well as real-time monitoring combined with the tracking of goods and linking them to Cloud services.

There have already been several research projects that address the combination of different independent manufacturers to so-called virtual factories. Most of these research projects focus primarily on the business-side in general and on aspects like partner-finding and factory-building processes in special. However no proven tools or technologies exist in the market that provide the creation of virtual factories applying end-to-end integrated Information and Communication Technology (ICT). ADVENTURE is aiming to provide such tools and processes that will help to facilitate information exchange between factories and move beyond the boundaries of the individual enterprises involved. The collaborative manufacturing process will be optimised by enabling the integration of factory selection, forecasting, monitoring, and collaboration during runtime.

ADVENTURE builds on concepts and methods of Service-oriented Computing and benefits from the advancements in this field. The monitoring and governance of the collaborative processes will be supported by technologies from the Internet of Things such as wireless sensors. Existing tools and services that can be integrated will be considered during the development of the platform for ADVENTURE.

The increased degree of flexibility provided through ADVENTURE will benefit SMEs especially as it helps them to react quickly to changes and to participate in larger, cross-organizational manufacturing processes. Furthermore, ADVENTURE will help manufacturers in assessing the environmental friendliness of actual manufacturing processes and resulting products and services. Other objectives of ADVENTURE include research in areas such as service-based manufacturing processes, adaptive process management, process compliance, and end-to-end-integration of ICT solutions.

1.2 Deliverable Purpose, Scope and Context

The purpose of this deliverable is to accompany the second prototype implementation of task 6.1. As such, its main purpose is to briefly clarify the scope of the prototype and to give download and installation instructions for the component.

In comparison to D6.1.1 the prototype has evolved in the following dimensions:

- **Part 1** – **Developer Friendliness**
  Dual HTTP / Message Routing (T4.4) interface that allows other component developers to interact with the SPE via XMPP, and check the http browsable version with the browser for changes. Furthermore explorable documentation is provided that describes how to interact with live process instances at runtime.
- **Part *2 – ADVENTURE Specific Events***
  Additional events covering simulation and semantic events, such as sensor events.
- ***Part 3 – Script Tasks***
  *An ADVENTURE specific CPEE **(Cloud Process Execution Engine)** extension (standalone service) that allows **to execute** script tasks written in Javascript. This includes the possibility to post Javascript over a Restful API and query the execution status or the values of variables defined within the scripts.*
- **Part 4 - Instantiation Extension**
  A simple standalone service that provides a single operation to facilitate the creation of an instance from a certain process model, and immediately start it in either simulation or execution mode.
- ***Part 5 – Translation Extension***
  *An ADVENTURE specific CPEE extension (standalone service) that allows the translation of BPMN 2**.0** (including the specific ADVENTURE extensions) into the CPEE's internal execution language.*
- **Part 6 – Adaptation Extension**
  An ADVENTURE specific CPEE extension (standalone service) that monitors and allows/ denies manual adaptations to process instances.

This second prototype will be delivered in the form of a zip bundle, containing a set of standalone services:

- "*//cockpit*": Create process instances, run performance tests and observe the runtime behaviour of ADVENTURE instances. This is a HTML-based UI that communicates with the SPE.
- *"//overlay":* Browsable documentation of the completely exposed SPE API. The documentation is browsable per instance (as the available operations change per instance and instance state).
- "*//instantiation*": Create new ADVENTURE specific instances, and immediately start them in either "simulation" or "execution" mode.
- *"//spe":* A simple-to-use service that uses the CPEE (Cloud Process Execution Engine) library, to provide Smart Process Execution, plus ADVENTURE specific configuration and plugins (i.e. endpoints for js-handler, translation, simulation, correlation and adaptation-guard – see below).

- *"//js-handler":* Execute Javascript script tasks (modify process instance variables based on return values from gateways), or evaluate Javascript conditions (e.g. used for loops, exclusive branch handling, inclusive branch handling, parallel branch handling).
- *"//translation-handler"*: Translate BPMN 2.0 (http://www.omg.org/spec/BPMN/20100524/MODEL) models, as well as the ADVENTURE specific extensions that have been introduced in D5.1, into CPEE specific process structure trees (see [1]).
- *"//correlation-handler":* assign gateway messages (sent through the Message Routing (T4.4) component) to SPE instances, based on message summary fields (see D3.3, Figure 52).
- *"//adaptation-guard":* Monitor and allow / deny adaptations to process instances.

Also included but focus of a different deliverable:

- *"//simulation":* Focus of D5.2.

Also included but just for testing purposes:

- *"//monitoring-test":* Implementation of an example monitor to solve integration issues with T6.2.

1.3 Document Status

This document is listed in the DOW as PU, which means 'Public'.

1.4 Target Audience

The target audience of this document are the ADVENTURE development team as well as ADVENTURE user partners who are assessing how the Smart Process Execution component enables the execution of the business logic behind a Virtual Factory. It will also be interesting for members of other national and EU projects who want to utilize a lightweight, scalable and flexible Process Execution Engine to drive internal and external control flow.

1.5 Abbreviations and General Terms

A definition of general, common terms and roles related to the realization of ADVENTURE as well as a list of abbreviations is available in the supplementary document "Supplement: Abbreviations and General Terms" which is provided in addition to this deliverable.
Further information can be found at: http://www.fp7-adventure.eu/glossary/

[1] J. Mangler, G. Stuermer, and E. Schikuta, "Cloud Process Execution Engine-Evaluation of the Core Concepts," Arxiv preprint arXiv:1003.3330, 2010.

1.6 Document Structure

This deliverable is broken down into the following sections:

- **Section 1** provides an overview about ADVENTURE and this deliverable.
- **Section 2** shows the connection of the second Smart Process Engine (SPE) prototype to the other components and the importance of the finalized component.
- **Section 3** points out which requirements have to be taken into account when trying to use the prototype, and what has to be provided before using the prototype as a developer.
- **Section 4** describes the installation of the prototype.
- **Section 5** details what the prototype does, how the server works and how the course of action is when trying to use the prototype.
- **Section 6** provides the further changes that have to be implemented to advance to the next milestones.
- **Section 7** is a small summary of this deliverable.

2 Scope and Relationship

The Smart Process Execution (SPE) component realizes the central point of control for the business logic that comprises a Virtual Factory. Figure 1 show the components covered by this deliverable in relation to ADVENTURE Architecture as elaborated in (D3.3). Process Execution and Adaptation are different (independent) components as the ADVENTURE partners expressed the wish to possibly exchange the Execution Engine, but keep Adaptation related functionality, as not many engines today support adaptation.

How this separation works is explained in more detail in the following sections. Adaptation, Simulation, Monitoring and many sub-components of Process Execution are interacting purely through an event-based interface and together from the Smart Process Execution (SPE).

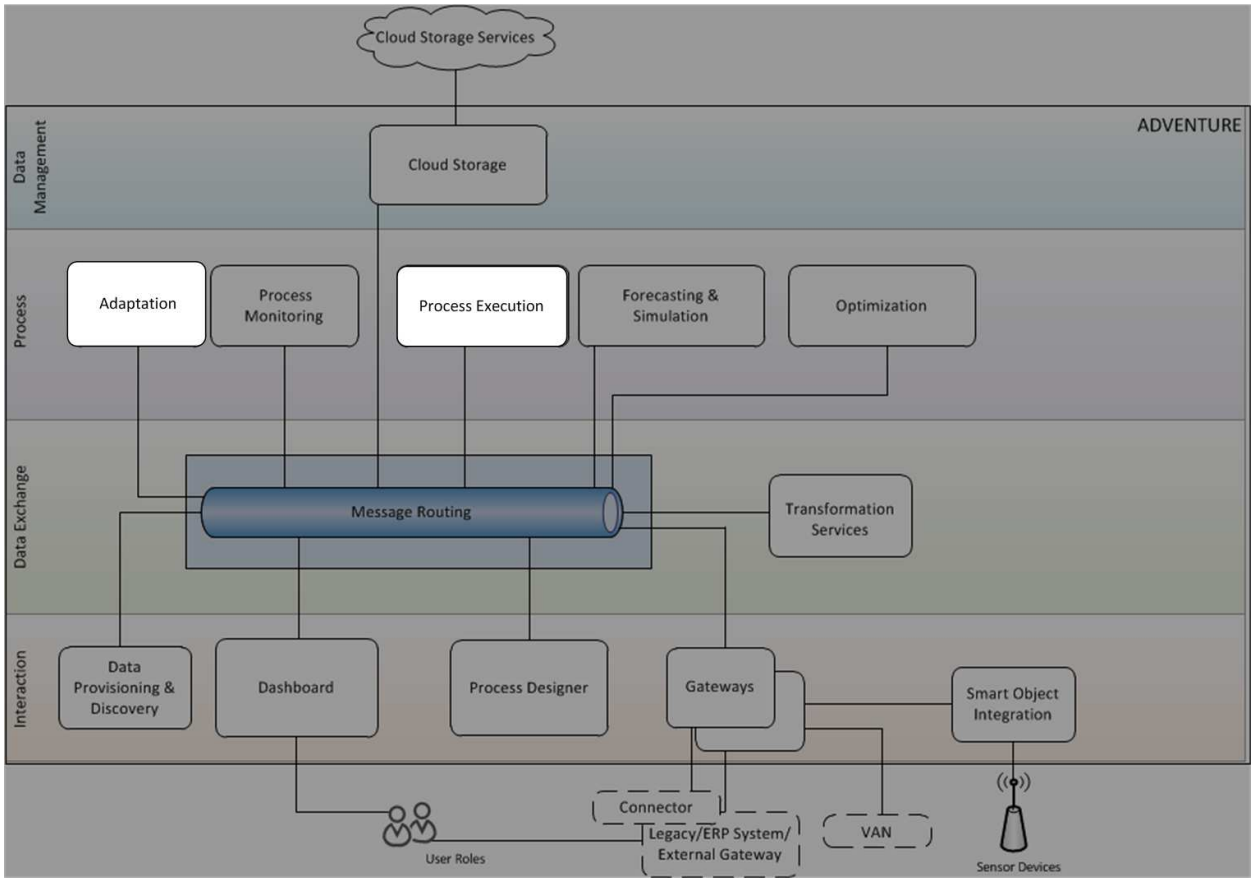

Figure 1: ADVENTURE Architecture

Figure 2 presents an architectural overview of the second SPE prototype (including Process Execution and Adaptation), which highlights the different core functions and how they interact with extensions for customization and developer friendliness.

The utilized Cloud Process Execution Engine (CPEE) library mainly consists of three classes:

- **CPEE::implementation**, which is responsible for execution and can be configured (via *riddl_opts* attributes to expose a certain set of events, protocol handler or execution language translation properties (i.e. ADVENTURE BPMN 2.0 translation to executable code, standard BPMN translation to executable code, BPEL translation, ...).
- **CPEE::HandlerWrapperBase** exposes an interface that has to be implemented by servers that run a certain *CPEE::implementation* configuration. It allows a developer based on execution, implement its own event semantics and protocol handling (in the case of ADVENTURE XMPP).
- **CPEE::Controller** exposes a set of methods that can be used in *SPE::HandlerWrapper* to send events. It exposes an event interface to all connected components, such as Adaptation.

All other components in Figure 2, are either used in the custom **SPE::HandlerWrapper**, or interact with the SPE through the dispersal of events (CPEE::Controller interface), which are also implemented and triggered in **SPE::HandlerWrapper.**

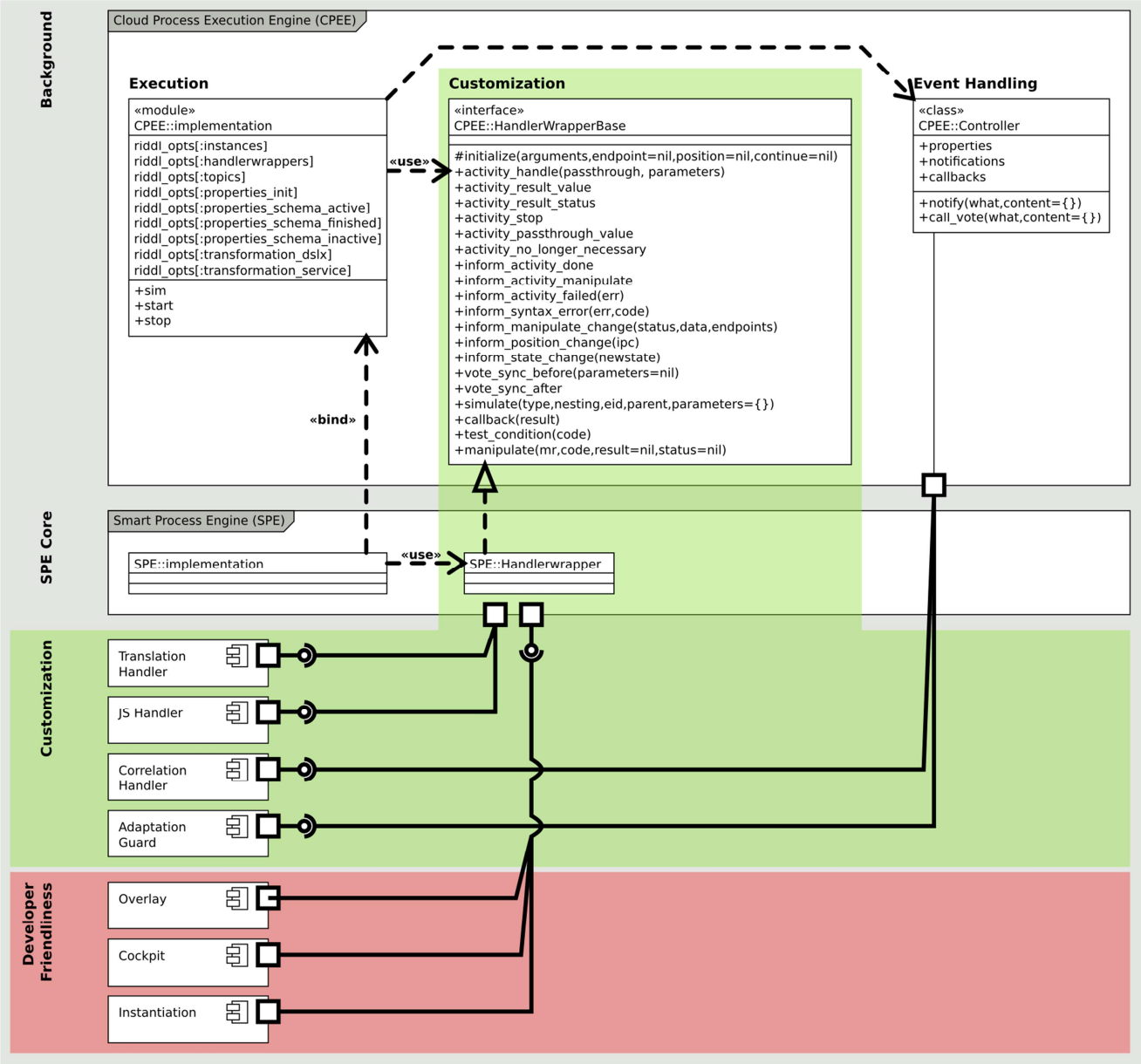

Figure 2: Scope of Prototype II

The Smart Process Execution prototype is especially important for executing Smart Processes and therewith realizing Virtual Factories. If further affects the following components:

Process Designer: Enable execution of processes by translating from BPMN to internal execution language. Each task in a process model may include Javascript code to:

- Modify process instance variables, based on the result from a call to a gateway.
- Reorder the list based on the results of a call to a gateway for a given list of potential partners that may handle the execution of a task.

- Trigger events for e.g. Monitoring or Adaptation based on the results of a call to a gateway.
- See Figure 2 – the "Instantiation" component invokes all necessary steps in the Smart Process Engine (SPE) that are necessary to (1) instantiate a process model and (2) start the instance (in either simulation or execution mode).

Gateways: their purpose is to encapsulate legacy systems of partners that participate in virtual factories. From the POV of SPE the following interaction takes place:

- Collect, transform and manage data from partners.
- Forward information from partners to other partners (through the gateways).
- See Figure 2 – the implementation in "SPE::Handlerwrapper" directly calls gateways according to the information provided with each task (i.e. endpoint of gateway, information about operation and parameters).
- See Figure 2 – the implementation in "SPE::Handlerwrapper" utilizes the mechanisms provided by "CPEE::Controller" to send events ("notify") to "Correlation Handler" component, whenever an instance is waiting for asynchronous information.

Monitoring: the purpose of monitoring is to collect all events that occur while a virtual factory in created and running. From the POV of SPE the following interaction takes place:

- Send syntactic and semantic events generated during process execution.
- See Figure 2 – the implementation in "SPE::Handlerwrapper" utilizes the mechanisms provided by "CPEE::Controller" to send a set of events ("notify") which are caught by the Monitoring component.

Forecasting & Simulation: the purpose of this component is to collect and aggregate information from partners (by interaction through gateways) about future capacity and behaviour of their manufacturing resources.

- Simulation is a special mode in the SPE.
- See Figure 2 – operation "sim" in module "CPEE::implementation" (exposed as a state through the REST XMPP interface) is used to start simulation mode.
- See Figure 2 – the interface "HandlerWrapperBase" provides the possibility to implement "simulate" which is called for each task during a simulation. The implementation in "SPE::Handlerwrapper" utilizes the mechanisms provided by "CPEE::Controller" to send events ("notify") which are caught by the Simulation component.

3 Requirements & Preparations

3.1 For Users

Users will not directly use the prototype, as it is an internal component only used by other ADVENTURE components.

3.2 For Developers

This version of the prototype is exposed in dual form:

- Through the Message Routing (T4.4) component at `process-execution@fp7-adventure.eu`. This part represents the integration with the messaging prototype, which is described in D4.4.1 (Message Routing Prototype).
- As a RESTful web service available under `http://fp7-adventure.eu/components/spe/cockpit` in order to allow simple performance and unit testing independent of the Message Routing Prototype.
  As an interactive documentation service, available under `http://fp7-adventure.eu/components/spe/overlay`.

In order to provide a good experience for developers, in conjunction with T4.4 (D4.4.1) a 1:1 translation from HTTP to XMPP calls has been developed. This especially helps in T6.4 (Integration) as developers can interact with HTTP based REST services to the XMPP solution developed in ADVENTURE.

4 Installation and Execution

For the ADVENTURE consortium a hosted version of the prototype is readily available using the Message Routing Library (see prototype D4.4.1) to contact:

`xmpp://adventureprocess-execution@fp7-adventure.eu`.

The following installation steps are only necessary for local installation (explained on the basis of an up-to-date Linux/Ubuntu 13.04) on an own server or client[2]

- Installation of an infrastructure for execution the process engine and components code:
  - `sudo apt-get install build essential libxml2-dev libxslt-dev libz-dev ruby ruby-dev libssl-dev git`
- Installation of the process engine libraries and dependencies:
  - `sudo gem install cpee`
- Clone the ADVENTUE SPE repository:
  - `sudo git clone http://donatello.wst.univie.ac.at/repositories/149/SPE/`
- Start infrastructure:
  - `SPE/spe/server start`
  - `SPE/instantiation/server start`
  - `SPE/correlation-handler/server start`
  - `SPE/translation-handler/server start`
  - `SPE/adaptation-guard/server start`
  - `SPE/js-handler/server start`
  - `SPE/overlay/server start`
- Make cockpit available in web server:
  - `sudo ln –s `pwd`/SPE/cockpit /var/www/cockpit`

After starting everything, a means to create and test process instances (independently of the Process Designer) is available under `http://localhost/cockpit`. Under `http://localhost:9297/` all technical documentation for interaction with the SPE itself can then be browsed.

[2] Be aware that these instructions override the instructions from D6.1.1.

5 Execution & Usage

All technical information for how to interact with the SPE itself is available under:

`http://fp7-adventure.eu/components/spe/overlay`

In this chapter *specific information about aforementioned sub-components is provided. In order to exemplify various aspects of process execution tasks and modelling elements of the processes depicted in* Figure 3 *and* Figure 4 *are used.*

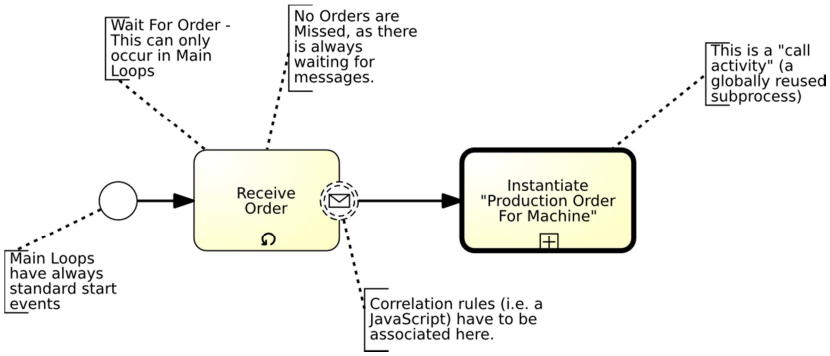

Figure 3: Main Loop Example Using Generic BPMN 2.0

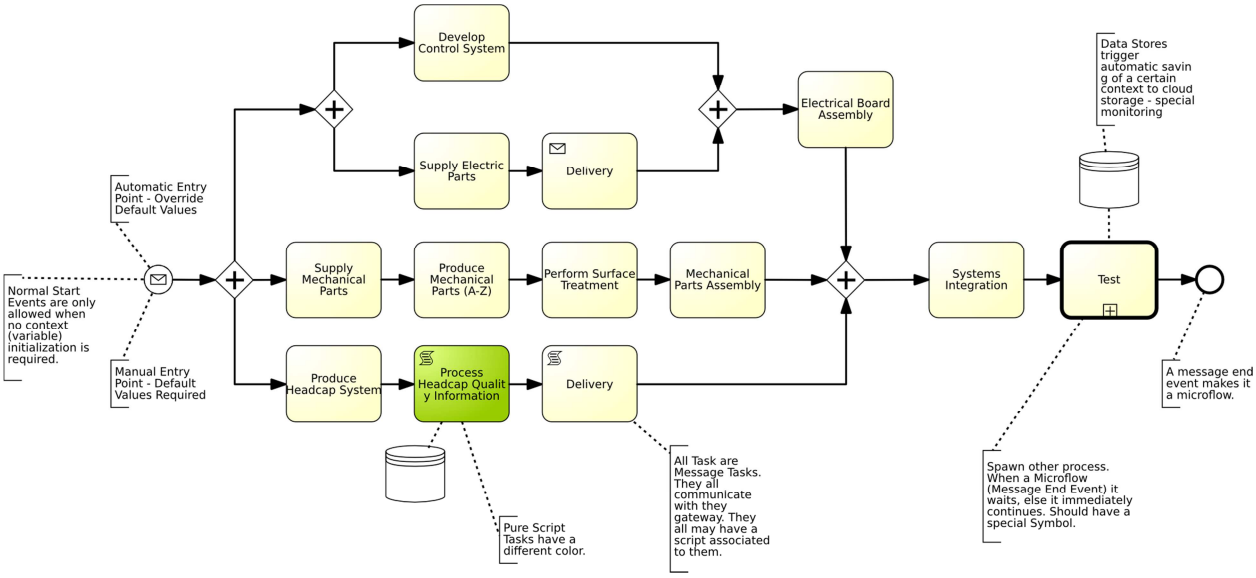

Figure 4: User Partner Process Example Using Generic BPMN 2.0

5.1 Instantiation

The purpose of the Instantiation service is to simplify the creation of process instances. The creation of instances (without the Instantiation service) on the SPE typically involves 5 steps:

- POST a new instance to / (returns {id})
- PUT the Virtual Factory id and process model to /{id}/properties/values/info
- PUT a process model to /{id}/properties/values/description

- POST subscriptions to event streams to /{id}/notifications/subscriptions
  - ADVENTURE Monitoring (T6.2) subscribes to a set of structural events, which occur when tasks are executed, as well as to state changes of the process instance itself (i.e. when the engine stops, starts or is ready). Sensor changes are semantic events that occur whenever a gateway returns a value (in the summary) that is marked as sensory data.

```
<handler url="xmpp://adventure_processmonitoring@fp7-adventure.eu">
  <events topic='properties/state'>change</events>
  <events topic='properties/datalements'>change</events>
  <events topic='properties/position'>change</events>
  <events topic='running'>
    activity_calling,activity_failed,activity_manipulating,activity_done
  </events>
  <events topic='sensor'>change</events>
</handler>
```

  - A Dashboard widget (e.g. one that displays a list of running instances per Virtual Factory, see T6.3) may subscribe to the same events as the monitoring, but processes them in a different way for display purposes.

```
<handler url="xmpp://adventure_gadget_ptio@fp7-adventure.eu">
  <events topic='properties/state'>change</events>
  <events topic='properties/position'>change</events>
  <events topic='running'>
    activity_calling,activity_failed,activity_manipulating,activity_done
  </events>
</handler>
```

  - The ADVENTURE Simulation (T5.2) component only listens to simulation events.

```
<handler url="xmpp://adventure_simulation@fp7-adventure.eu">
  <events topic='simulating'>step</events>
</handler>
```

  - The Correlation component (part of T6.1) only listens to correlation events.

```
<handler url="xmpp://adventure_correlation@fp7-adventure.eu">
  <events topic='correlation'>request</events>
</handler>
```

  - The Adaptation Guard (part of T6.1), that monitors and allows or denies adaptations to process instances, only listens to events that are fired when the process model changes. “Votes” are special events that allow the Adaptation Guard to DENY the change.

```
<handler url="xmpp://adventure_adaptationguard@fp7-adventure.eu">
  <votes topic='properties/description'>modify</votes>
</handler>
```

- PUT ‘ready’ to /{id}/properties/values/state to signal the creation of a new instance to the Monitoring component (and other components).
- PUT ‘simulating’ or ‘running’ to /{id}/properties/values/state to start the instance in the respective mode.

The following interface description describes the input and output in order to trigger the above described functionality.

Message	POST	
URL	/	
Input	Xml	MIMETYPE: */xml
	targetmode	PARAMETER: value ‘ready’ | ‘simulating’ | ‘running’
Output	id	PARAMETER: integer

After executing the POST operation, the SPE returns an integer id that denotes the id of the instance, that has been created in the SPE. Be aware that while this service is available under `xmpp://adventure_processinstantiation@fp7-adventure.eu`, the instance itself will be available under `xmpp://adventure_processexecution@fp7-adventure.eu/{id}`.

The following operation, available under `xmpp://adventure_processinstantiation@fp7-adventure.eu/{id}`, returns information about where the real instance is hosted (i.e. `xmpp://adventure_processexecution@fp7-adventure.eu/{id}`), and requests a redirect.

Message	*GET*	
URL	/{id} (see output parameter of operation POST in /)	
Input	NONE	
Output	Location	HEADER
	Info	MIMETYPE: application/xml
	HTTP/XMPP Status 302 (redirect to equivalent SPE instance)	

5.2 Correlation Handler

The purpose of the Correlation Handler is to assign messages that have no particular recipient instances to one or more instances. In order to do so, the Process Designer (D5.1.2) contains elements and attributes to express the desire to process certain messages that contain certain pieces of data.

Examples for such elements can be seen in Figure 3 and Figure 4 – “Boundary Non-interrupting Message Events” and “Message Tasks” (tasks with envelope symbol top left).

- Figure 3 is a Main Loop (as described in D3.3) with the means to start and stop Virtual Factories. The boundary non-interrupting message element has to contain the correlation rules. Whenever a message arrives, the “Receive Order” task, that infinitely waits for orders, executes the sequence that contains “Instantiate ’Production Order For Machine’”[3].
- In Figure 4, the task “Delivery” waits for the asynchronous arrival of a delivery message.

[3] Side note: this task in turn calls the „Instantiation“, described in 5.1.

The Correlation Handler interface-wise is a standard event receiver. It receives messages from the SPE. The messages are triggered in the custom “SPE::HandlerWrapper” as shown in Figure 2, based on the type of the task.

Thus the POST to the correlation Service follows the standard structure of an event receiver. “Key” denotes the subscription id, topic in this case will always be “correlation”, event will always be “request”. “notification” contains the important information:

- “**name**” of the element in the summary of a gateway message.
- Correlation “**type**”. Three types are supported:
  - “match” – the whole contents of the summary element has to match the correlation criterion.
  - “xpath” – it is assumed that the summary is a piece of xml, the correlation can be determined by an xpath. If the xpath returns “”, “false”, or an empty list of nodes, no correlation will be given.
  - “rest” – the contents of the summary element, together with the correlation criterion is sent (via POST) to an external XMPP service, which returns “true” or “false”.
- Correlation “**criterion**” – depends on the type. For “match” it is a string, for “xpath” an xpath expression, for “rest” an url to the functionality that can verify whether the criterion correlates.
- “instance_id” denotes the instance for which the correlation rule is registered.
- “task_id” denotes the task id (in this certain instance) for which the correlation rule is registered.

Finally “fingerprint” contains a hash of all elements, to ensure that the event has not been tampered with.

Message	POST	
URL	/	
Input	key	PARAMETER: string
	topic	PARAMETER: string, PATTERN: [\w_\/]+
	event	PARAMETER: string, PATTERN: [\w_]+
	notification	PARAMETER: string
	fingerprint	PARAMETER: string
Output	id	PARAMETER: integer

Upon sending an event, an “id” is returned that can be used to find the registered rule, as well as delete it. Please note that correlation rules that require checking several fields can be submitted as singe rules. An arbitrary number of name/type/criterion triplets can be supplied with each rule. In order to correlate DIFFERENT messages for the same tasks, several correlation rules for the same task may be registered.

Below the two ways of accessing the registered rules are documented: (1) through the id obtained when creating a rule, and (2) through using the instance and task id's a rule is associated with.

Message	GET	
URL	/{id} or /{instance_id}/{task_id} (see output parameter of operation POST in / as well as description above)	
Input	NONE	
Output	Name	PARAMETER: string
	Type	PARAMETER: value 'match', 'xpath', 'rest'
	criterion	PARAMETER: string (e.g. 'value', '/test/value[text()="test"]', 'xmpp://adventure_correlationtester@fp7-adventure.eu')
	…	(arbitrary number of name/value pairs, i.e. endpoints, the combination of them has to be present in one message)

As expected the operation GET returns the triplets that the rule consists of. Finally the operation DELETE gets rid of the correlation rule. DELETE is always called in "SPE::HandlerWrapper", whenever:

- The process instance state changes (e.g. to stopped)
- The task that a correlation rule is associated with is finished.

Thus for the Main Loop depicted in Figure 3, the correlation rule is registered when the instance starts (i.e. only one instance of the Main Loop is allowed), and is unregistered when the instance stops. For the task "Delivery" in Figure 4, multiple registrations may exist, one for each running process instance.

Message	DELETE
URL	/{id} (see output parameter of operation POST in /)
Input	NONE
Output	HTTP Status 200 (successful delete correlation rule)

5.3 Translation Handler

The purpose of the translation handler is to translate the BPMN model to an executable process structure tree, as described in [MSS10]. This functionality is triggered through the custom "SPE::HandlerWrapper" as depicted in Figure 2. The service interface of the Translation covers three cases.

First: the extraction of the process model from the BPMN file.

Message	POST	
URL	/	
Input	description	MIMETYPE: text/xml
	type	PARAMETER: value ‘description’
Output	description	MIMETYPE: text/xml

Second: the extraction of all data-elements from the BPMN process model. This is a separate step, as we might come up with different notions of what a data-element is. E.g. a standalone (not connected to the process) sensor element may automatically lead to a sensor related data-element that always contains the last value.

Message	POST	
URL	/	
Input	description	MIMETYPE: text/xml
	type	PARAMETER: value ‘dataelements’
Output	name	PARAMETER: string
	value	PARAMETER: string
	…	(arbitrary number of name/value pairs, i.e. data elements)

Third: the extraction of all endpoints (gateway addresses) from the process model.

Message	POST	
URL	/	
Input	description	MIMETYPE: text/xml
	type	PARAMETER: value ‘endpoints’
Output	name	PARAMETER: string
	value	PARAMETER: string
	…	(arbitrary number of name/value pairs, i.e. endpoints)

5.4 Adaptation Guard

The adaptation guard accepts or denies changes to the process model based on an event (see Section 5.1) that is sent whenever an instance (i.e. the model of an instance) is changed. The Adaptation Guard implements the concepts found in [SRD04]. The Adaptation Guard considers the progress of execution when allowing or denying adaptations to process models. The Adaptation Guard may be subject to heavy change, due to integration work with the Azevedos use-case which may require more relaxed criteria, and due to the fact that changes may be possible when considering that when changing the process structure AND gateways (i.e.

business partners associated with tasks), changes may be valid despite violation of correctness criteria.

5.5 Javascript Handler

The Javascript handler is being exposed as only a RESTful API (due to the lack of working Javascript XMPP libraries), and is utilized in “SPE::HandlerWrapper”, whenever:

- A script is associated with a task.
- A condition specified in a loop or decision element has to be evaluated.

New tasks to be executed can be submitted to the engine via a POST message, while the execution can be handled synchronously and asynchronously. In the synchronous mode, after the submitted java script code has been executed, as a response to the post call the id of the newly created execution object is returned.
In the asynchronous mode, immediately after a post message has been received, an id that identifies the newly created execution object will be returned. After the posted script task has then been executed, a status message concerning whether the posted script task was executed successfully or not is being sent to the delivered “callback” url. In both modes, synchronous and asynchronous, at all times the execution status can be retrieved using GET queries.

The restful API, over which Javascript tasks can be submitted via a POST method, is available under `http://fp7-adventure.eu:9292/.`

Javascript code can be posted using the following parameters:

- “code” (mandatory): A String containing the Javascript code to be executed.
- “dataelements” (mandatory): a JSON object, containing data elements that need to be accessed during the execution of the posted Javascript code.
- “endpoints” (mandatory): a JSON object, containing endpoints that need to be accessed during the execution of the posted Javascript code.
- “callback” (optional): an optional String, containing a URL. If the callback parameter is defined, the script task handler will be executed asynchronously.
- “resultname” (optional): the name of a variable in which can be used to store execution results.
- “resultvalue” (optional): the pre-defined value of the result variable.

The following table gives an overview over the input and output parameters of the POST service that can be called to post new Javascript tasks.

Message	POST	
URL	/	
Input	code	PARAMETER: string
	dataelements	MIMETYPE: application/json
	endpoints	MIMETYPE: application/json
	callback	PARAMETER: string
	resultname	PARAMETER: string
	resultvalue	PARAMETER: string
Output	id	PARAMETER: string

After Javascript code has been posted using the Restful API, various GET-functions are being offered in order to query different parameters of the execution object. These queries can be made at all times, even while the posted Javascript code is being executed. The following GET-functions are being offered:

- /{id}: Return all information related to the execution object of {id}.
- /{id}/dataelements: Return the current values of datalelements.
- /{id}/endpoints: Return the current values of endpoints.
- /{id}/status: The result of the execution of the Javascript code.
- /{id}/state: Information regarding whether the Javascript code has already been executed, or has not yet been executed/is currently being executed.

The following tables give an overview over the input and output parameters of the GET methods of the service that can be called to retrieve information concerning the execution status and results of posted Javascript tasks.

Message	GET	
URL	/{id}	
Input		
Output	dataelements	MIMETYPE: application/json
	endpoints	MIMETYPE: application/json
	status	PARAMETER: string
	state	PARAMETER: string

Message	GET	
URL	/{id}/{element}	
Input		
Output	element	MIMETYPE: application/json

Furthermore, an execution object can be deleted by calling the DELETE method on /{id}.

Message	DELETE
URL	/{id}
Input	
Output	HTTP Status 200 (Instance successfully deleted) HTTP Status 500 (Instance does not exist)

In order to be able to test the behaviour of the Javascript handler component, several unit tests have been developed which can be run by 'make test' in the respective subdirectory (`SPE/js-handler/`).

5.6 Further Components

The "Overlay" component mentioned above allows developers to explore live process instances and read documentation about how to e.g. stop and start instances and apply modifications, or subscribe to an events screen at runtime. In order to not clutter this document with too much technical information, the exploration of `http://fp7-adventure.eu/components/spe/overlay` is encouraged.

6 Limitations & Further developments

Although this deliverable constitutes the final prototype based on the planning in the DOW, further adaptations might become necessary due to:

- Still changing correlation requirements, as final Forecasting and Simulation (D5.2) and Smart Object integration (D5.3) prototypes are not yet due, and are not fully integrated yet.
- Still changing adaptation requirements, as the final Monitoring (D6.2) prototype that handles the triggering of alerts and (possibly) automatic adaptation, is not yet due, and not fully integrated yet.
- Still changing translation requirements, because the final Smart Object integration (D5.3) prototype is not due yet, and integrating the component may lead to adaptations how to translate custom ADVENTURE BPMN 2.0 to code executable by the SPE.

The current prototype is functionally complete. Due to the nature of the project which provides a schedule for the integration with user partners, sensor information integration, simulation and forecasting as well as optimization in the third year, the prototype in its current stage will see an integration with the aforementioned components, with refinements to components as mentioned in Executive Summary.

It is furthermore planned to add more sensor related events, which will facilitate the Monitoring component in providing information to ADVENTURE Brokers.

Finally it will be necessary to test the current prototype in scenarios, jointly developed with user partners, and possibly derive an automated test suite.

7 Summary

This document shows how the second Smart Process Execution prototype can be installed and provides insight how to test its capabilities.. This document explains how the first Smart Process Execution prototype has been expanded with extensions for developer friendliness, the support for ADVENTURE specific events, scripts, instantiations, and the translation of BPMN 2.0 models to the internal execution language and an adaptation extension. Although the second prototype is functionally complete, it will see further developments due to the need for integration with user partners, sensor information, simulation and forecasting as well as optimization.

8 References

[MSS10] J. Mangler, G. Stuermer, and E. Schikuta, “Cloud Process Execution Engine-Evaluation of the Core Concepts,” Arxiv preprint arXiv:1003.3330, 2010.

[RRD04] S. Rinderle, M. Reichert, and P. Dadam, “Correctness Criteria for Dynamic Changes in Workflow Systems – A Survey.,” Data and Knowledge Engineering, vol. 50, no. 1, pp. 9–34, 2004.