OPAALS PROJECT

Contract n° IST-034824

WP2: Automatic Code Structure and Workflow Generation from Models

Del2.5 - Extended automated code/workflow generation example for one real-world application case

Project funded by the European Community under the "Information Society Technology" Programme

Contract Number: IST-034824

Project Acronym: OPAALS

Deliverable N°: D2.5

Due date: March, 2009 (M34)

Delivery Date: 15 June 2009

Short Description: In this report we describe the application of the conceptual outcomes and software tools obtained in the previous research activities (D2.2, D2.3, D2.4) in the field of dynamic workflow reconfiguration. The document shows a real case scenario of dynamic workflow reconfiguration by means of a natural language based specification.

Author: T6 ECO (Antonella Filieri, Antonio Margarito), SUAS (Thomas Kurz, Raimund Eder)

Partners contributed: T6 ECO, SUAS

Made available to: OPAALS Consortium and European Commission

Versioning		
Version	**Date**	**Name, organization**
0.1	20/5/09	T6 ECO, SUAS
0.2	15/6/09	T6 ECO, SUAS

Quality check

Internal Reviewers: Paul Krause, TV Prabhakar

Dependences:

Achievements*	In the reported period T6 ECO developed: - A deep analysis of different types of workflow dynamic reconfiguration cases in the enterprise domain to identify a concrete example that can be also very representative of the today's importance of the subject; - a continue analysis of the open source workflow engine solutions searching for new features/functionalities in order to evaluate the possibility to enhance workflow reconfiguration and simplify the dynamic workflow reconfiguration; - a rationalization of the prototype module previously implemented, which is able to perform a comparison between two versions of process definition model in order to distinguish between admissible or not-admissible instance modifications. In the reported period T6 ECO did not develop: - a complete solution for the implementation of a dynamic workflow reconfiguration due to some critical aspects related to the lack of customization features of the modern open source workflow engines.
Work Packages	This deliverable is connected to activities of WP3 that deals with evolutionary networks of SMEs. Such scenario is characterized by the necessity of handling inter-organizational re-configuration of workflows in order to react to environmental variations.
Partners	University of Surrey (UniS), Indian Institute of Technology Kanpur (IITK), University of Kassel (UniKassel), University of Central England Birmingham (UCE).
Domains	Computer science domain.
Targets	Computer Science researchers, SMEs and public administrations.
Publications*	The reported work has still not been published.
PhD Students*	

Outstanding features*	With this report we have enriched our knowledge about workflow reconfiguration. Thanks to the previously done precise analysis of workflow reconfiguration requirements and with an attempt to implement them, we are able to make a positive contribution to the state of art. Moreover the work done could have a significant impact to how effectively and practically support the changes of a business process model with positive effects for SMEs.
Disciplinary domains of authors*	Antonella Filieri: Computer Science and Business Modelling Domain Antonio Margarito: Computer Science and Business Modelling Domain Thomas Kurz: Information Technologies, Software and Systems Engineering, Interpersonal Communication Raimund Eder: Information Technologies, Software and Systems Engineering

The information marked with an asterisk (*) is provided in order to address Recommendation n. 4 from the Year 2 review report

Table of contents

Figures and Tables

1 Introduction

This report shows a simple case study in which a real-world workflow is dynamically reconfigured to meet a business requirement at first, and to optimize the performance of the process at a later moment.

Dynamic workflow reconfiguration for company workflow is done based on the studies conducted and published earlier (see OPAALS Deliverable D2.4 [4]) that are essential knowledge for understanding this work. This report highlights how it is possible to leverage the previous studies and implemented tools to manage the difficult problem of dynamic reconfiguration of workflows.

The scenario takes into account an enterprise in the mechanical manufacturing sector. The enterprise, whose name is not mentioned for privacy and security reasons, has its core business in the maintenance of big and complex machines. The organizational unit that has the responsibility for the maintenance process must issue invoices to customers for each performed maintenance activity. The rest of the document is organized as follows. Section 2 introduces the process in its “as is” status, providing both a BPMN [8] and an SBVR [9] representation, as described in OPAALS Deliverable D2.2 [2] and D2.3 [3]. Section 3 describes the two-step reconfiguration of the considered process, explaining the motivations for the modifications and taking into account the possible problems that such modifications may encounter on the existing workflow instances. Finally Section 3 reports the results of the reconfiguration execution on the real-world scenario.

2 Scenario Description (textual description)

In this section the scenario to be reconfigured is described through a textual description and both a BPMN [8] and an SBVR [9] representation.

2.1 Process description (BPMN description)

Every time the company performs maintenance operations on behalf of a customer, if a payment is due, then an invoice is generated. The information that an invoice is composed of are divided into descriptive and tabular data. The former contain all data related to the company (i.e. company name, address, VAT number, etc), to the customer (i.e. customer name, address, SSN, VAT number, etc) who commissioned the intervention for which the invoice is issued, and the terms of payment. The latter refers to the total cost of the maintenance operations and to its details in terms of atomic operations and VAT.

The company prepares the invoice and then sends it to the customer. The customer analyzes and validates the invoice entries regarding the type of the performed operations.

Once the analysis phase is completed, the client sends the request for payment. The client waits until the funds necessary to pay the bill arrive, and then he creates the payment order and sends to the company a payment communication. This communication allows the company to collect the total invoice amount.

If the customer verifies, during the invoice analysis phase, that there are irregularities, penalties will be generated. This situation may delay the invoice payment process because the penalty must be communicated to the company that will take care of its management.

To manage possible penalties, 7 days are at most awaited before starting to collecting the whole invoice amount. Moreover a communication is sent to the client so that he can manage the monetary penalty. The subsequent steps are the same as we previously described.

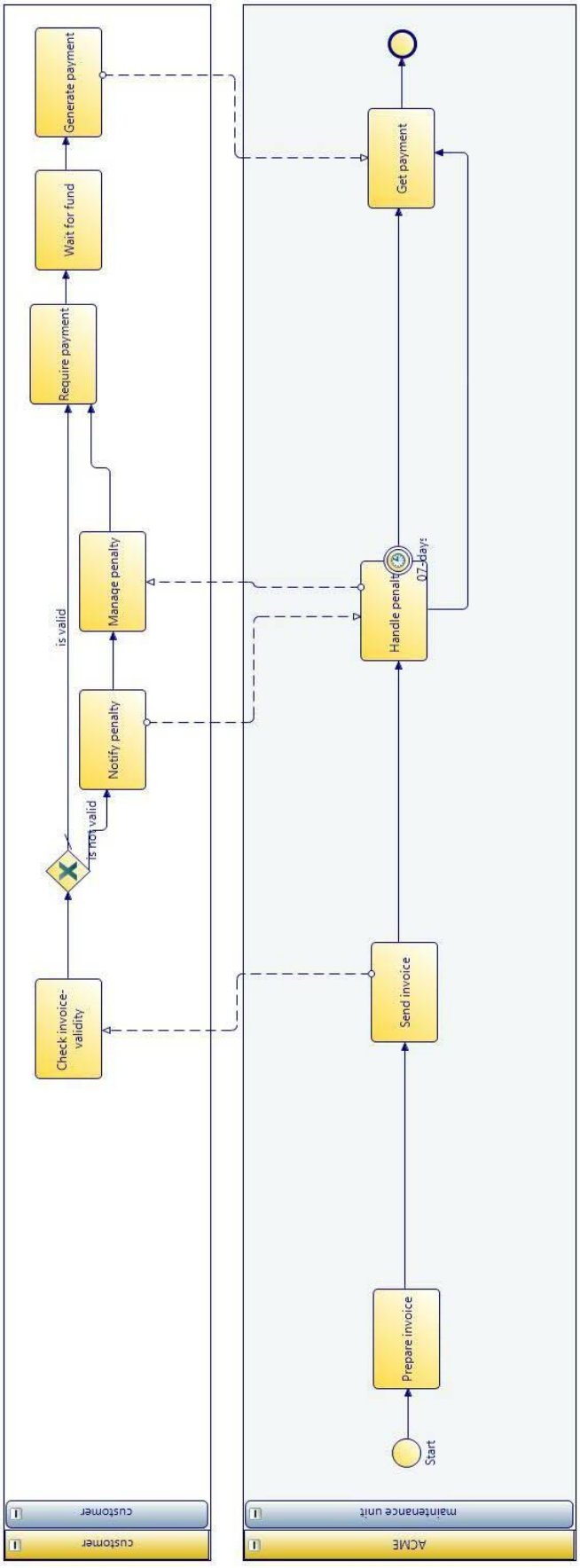

Figure 1 – As Is Business Process Diagram

2.2 SBVR Representation of the Process Model (SBVR)

In this section we will provide a detailed description of the SBVR Structured English representation of the billing process example. This description allows business people to describe their business in a language that is very near to the language they use every day. The approach refers to the results of OPAALS Deliverable D2.2 and D2.3 [2] [3] in which our research team described how to represent simple workflows through SBVR Structured English [9].

The first step is the definition of the process vocabulary (or the reuse of an existing one) made up of terms and fact types.

The SBVR SE Vocabulary for the billing process example is shown in Listing 1.

Vocabulary

Maintenance-Unit

Customer

Invoice

Invoice-validity

Penalty

Payment

Fund

Maintenance-Unit *prepares* invoice

Maintenance-Unit *sends* invoice

Customer *checks* invoice-validity

Invoice *is valid*

Customer *notifies* penalty

Customer *manages* penalty

Customer *requires* payment

Maintenance-Unit *handles* penalty

Customer *waits for* fund

Customer *generates* payment

Maintenance-Unit *handles* penalty

Maintenance-Unit *gets* payment

Maintenance-Unit *is role of* ACME

Listing 1 – Billing Process SBVR SE Vocabulary

After the vocabulary has been specified, in the second step the ruleset is defined in order to model transitions and constraints that characterize the process.
The SBVR SE Ruleset for the billing process example is shown in Listing 2.

Rules

After Maintenance-Unit *prepares* the invoice then Maintenance-Unit *sends* the invoice.
After Maintenance-Unit *sends* the invoice then the customer *checks* the invoice-validity.
After the customer *checks* the invoice-validity if the invoice *is valid* then the customer *requires* the payment.
After the customer *checks* the invoice-validity if the invoice *is not valid* then the customer *notifies* the penalty.
After the customer *notifies* the penalty then the customer *manages* the penalty.
After the customer *notifies* the penalty then Maintenance-Unit *handles* the penalty.
After the Maintenance-Unit *handles* the penalty then the customer *manages* the penalty.
After the customer *manages* the penalty then the customer *requires* the payment.
After the customer *requires* the payment than the customer *waits for* funds.
After the customer *waits for* funds than the customer *generates* the payment.
After the customer *generates* the payment then the Maintenance-Unit *gets* the payment.
After the Maintenance-Unit *sends* the invoice then the Maintenance-Unit *handles* penalties.
After the Maintenance-Unit *handles* the penalty then the Maintenance-Unit *gets* the payment
While the Maintenance-Unit *handles* the penalty if the time-elapsed *is* 7 days then the Maintenance-Unit *gets* the payment.

Listing 2 – Billing Process SBVR SE Ruleset

3 Detailed Reconfiguration Scenario Description (textual description)

In this section we describe how the emergence of new business requirements leads to changes in the process model, and how such changes can be translated in the reconfiguration of a running instance.
The process model described in the previous section will be modified and the reconfiguration patterns will be derived from the new version of the model according to the theoretical studies made and using the notation introduced in the OPAALS Deliverable D2.4.

3.1 Business Requirement

The billing process must be changed because of new requirements from the customers. Before sending the invoice to the customer, indeed, the company is required to perform a “customer acceptance test” in the presence of the customer. After that, the customer can accept or reject the maintenance results. The insertion of this testing activity leads to the re-modelling of the initial part of the billing process as shown in Figure 2.

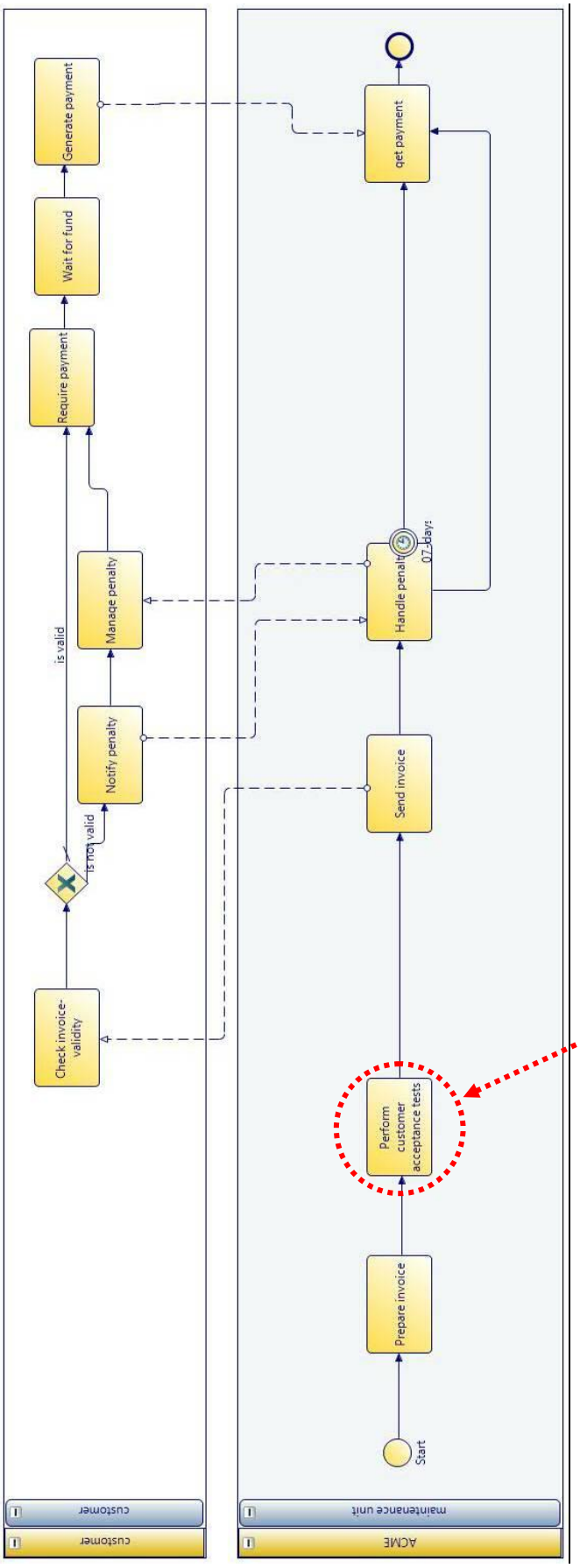

Figure 2 – Modified Process Diagram for business requirement

3.1.1 Variations in the SBVR representation of the process model

The newly introduced activity in the business process is reflected by the introduction of some new elements in the SBVR representation, namely a new term, a new fact type and a new business rule. In particular the added term is "customer-acceptance-test". Such term is used in the new fact type "Maintenance-Unit *performs* customer-acceptance-test" that represents the new activity. These changes are shown in Listing 3. The new business rule is "After the Maintenance-Unit *performs* the customer-acceptance-test then the Maintenance-Unit *sends* the invoice". Moreover the business rule "After the Maintenance-Unit *prepares* the invoice then the Maintenance-Unit *sends* the invoice" is changed in "After the Maintenance-Unit *prepares* the invoice then the Maintenance-Unit *performs* the customer-acceptance-test". These additions and changes are shown in Listing 4.

Vocabulary

Maintenance-Unit

Customer

Invoice

Invoice-validity

Penalty

Payment

Fund

customer-acceptance-test

Maintenance-Unit *prepares* invoice

Maintenance-Unit *performs* customer-acceptance-test

Maintenance-Unit *sends* invoice

Customer *checks* invoice-validity

Invoice *is valid*

Customer *notifies* penalty

Customer *manages* penalty

Customer *requires* payment

Maintenance-Unit *handles* penalty

Customer *waits for* fund

Customer *generates* payment

Added Term

Added Fact Type

Maintenance-Unit *handles* penalty

Maintenance-Unit *gets* payment

Maintenance-Unit *is role of* ACME

Listing 3 – Modified Billing Process SBVR SE Vocabulary

After the vocabulary is specified, the second stage consists in the definition of the ruleset, as shown in Listing 4.

Rules

Modified and added Rules

After the Maintenance-Unit *prepares* the invoice then the Maintenance-Unit *performs* the customer-acceptance-test.

After the Maintenance-Unit *performs* the customer-acceptance-test then the Maintenance-Unit *sends* the invoice

After Maintenance-Unit *sends* the invoice then the customer *checks* the invoice-validity.

After the customer *checks* the invoice-validity if the invoice *is valid* then the customer *requires* the payment.

After the customer *checks* the invoice-validity if the invoice *is not valid* then the customer *notifies* the penalty.

After the customer *notifies* the penalty then the customer *manages* the penalty.

After the customer *notifies* the penalty then Maintenance-Unit *handles* the penalty.

After the Maintenance-Unit *handles* the penalty then the customer *manages* the penalty.

After the customer *manages* the penalty then the customer *requires* the payment.

After the customer *requires* the payment than the customer *waits for* funds.

After the customer *waits for* funds than the customer *generates* the payment.

After the customer *generates* the payment then the Maintenance-Unit *gets* the payment.

After the Maintenance-Unit *sends* the invoice then the Maintenance-Unit *handles* penalties.

After the Maintenance-Unit *handles* the penalty then the Maintenance-Unit *gets* the payment.

While the Maintenance-Unit *handles* the penalty if the time-elapsed *is* 7 days then the Maintenance-Unit *gets* the payment.

Listing 4 – Modified Billing Process SBVR SE Ruleset

3.1.2 Considerations upon reconfiguration patterns and conditions

The modifications of a process model can be analyzed by decomposing (thus reducing its complexity) such modifications into elementary blocks, called reconfiguration patterns (see OPAALS Deliverable D2.4 [4]). This approach allows recognizing conditions and constraints on the reconfiguration of running instances of a process if its model is modified. The evaluation of the admissibility of the reconfiguration of a running process instance is thus based on the admissibility (on that instance) of each of the reconfiguration patterns the modification is composed of.
When a new process (or a new version of a process) is specified in SBVR SE, an ad-hoc developed library is used to translate it into an XPDL representation (see OPAALS Deliverable D2.2 [2] and Deliverable D2.3 [3]) and to evaluate its content.
First of all the name and the version of the model are compared to couples “name-version” of existing running instances (already deployed in the workflow engine). If the name does not match to any process instance, then the new process is deployed. If running instances with the same name exist in the workflow engine, then a deeper analysis is needed in order to evaluate the reconfiguration admissibility of each of those instances: the XPDL file of the new model is compared to the XPDL file of the running instances of the latest version of the process model in order to identify the required reconfiguration patterns. Once such patterns are found, information about the status of the running instances are used to determine the admissibility of their reconfiguration. If all reconfiguration patterns are admissible for a specific instance, then that instance is admissible for reconfiguration.
The pattern applied to the insertion of the required testing activity is the sequential insertion. So the criteria for migrating running instances of the process to the new configuration is the following: the *Sequential Insertion* reconfiguration pattern is an admissible operation only if all tasks that follow the insertion point are in the “initial” state. Therefore all the instances of the process in which the preparation of the bill is not already completed can be migrated to the new model so that it is possible to meet the new requirements.

3.2 Process refinement

Optimization possibilities for the new process are identified after that the process model has been changed for business requirements: the previously introduced activity (i.e. “Perform customer-acceptance-test”) does not depend on the preparation of the bill (i.e. “Prepare Invoice” task). Hence the execution time of the process can be improved by making parallel the two activities. The outcome of the optimization is a process model rearranged as in Figure 10.

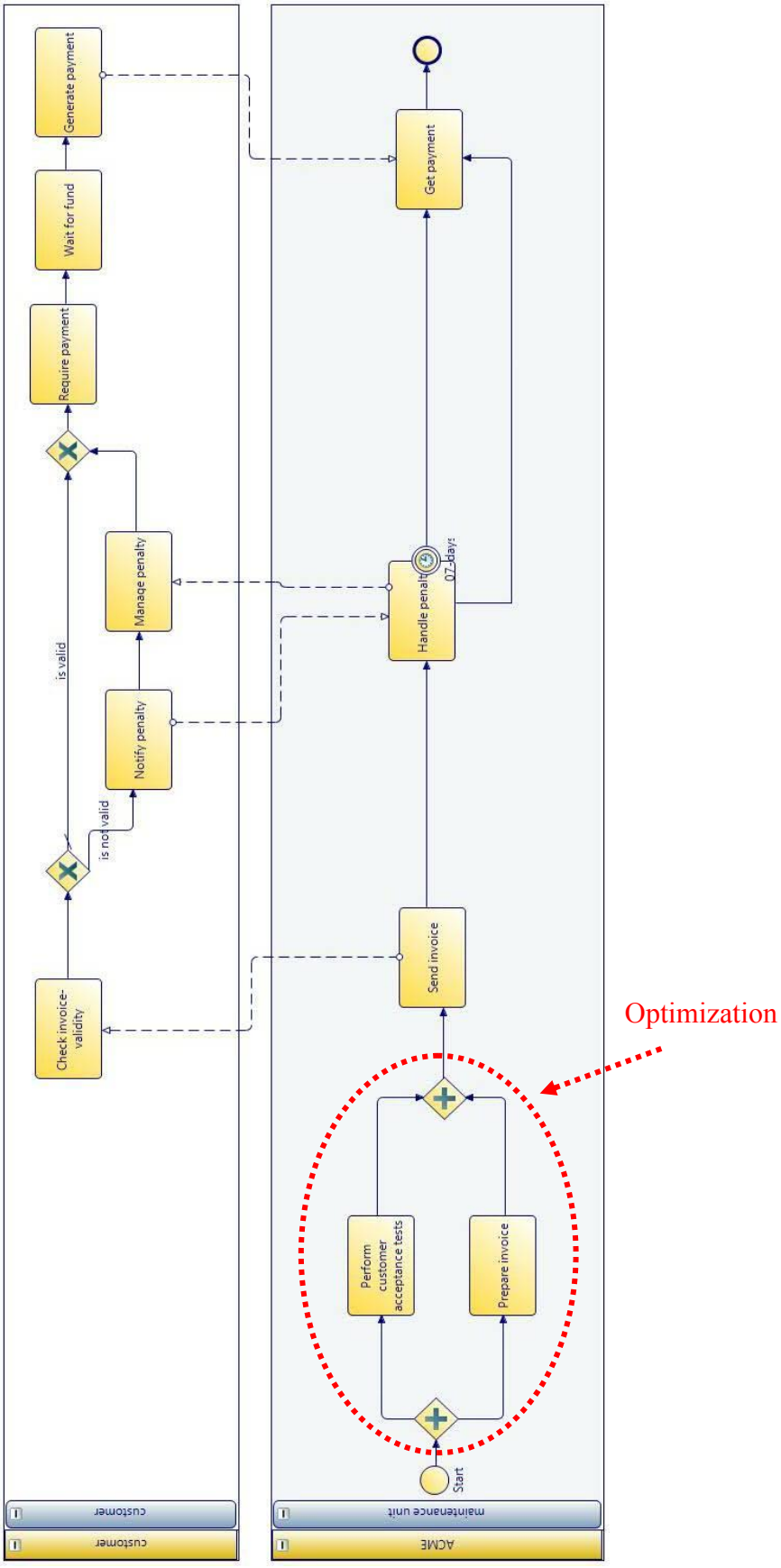

Figure 3 – Refined Business Process Diagram

3.2.1 Variations on the SBVR representation of the refined process model

The parallelization of the two activities results in a variation in the SBVR representation of the process. There is no change in the vocabulary, so the vocabulary remains the same showed in Listing 3, but one business rule is deleted and one new business rule is introduced. In particular the business rule "After the Maintenance-Unit *prepares* the invoice then the Maintenance-Unit *performs* the customer-acceptance-test" is no more needed, and the new business rule "After the Maintenance-Unit *prepares* the invoice then the Maintenance-Unit *sends* the invoice" is added. These modifications are shown in Listing 5.

Rules

After the Maintenance-Unit *prepares* the invoice then the Maintenance-Unit *sends* the invoice

After the Maintenance-Unit *performs* the customer-acceptance-test then the Maintenance-Unit *sends* the invoice

After Maintenance-Unit *sends* the invoice then the customer *checks* the invoice-validity.

After the customer *checks* the invoice-validity if the invoice *is valid* then the customer *requires* the payment.

After the customer *checks* the invoice-validity if the invoice *is not valid* then the customer *notifies* the penalty.

After the customer *notifies* the penalty then the customer *manages* the penalty.

After the customer *notifies* the penalty then Maintenance-Unit *handles* the penalty.

After the Maintenance-Unit *handles* the penalty then the customer *manages* the penalty.

After the customer *manages* the penalty then the customer *requires* the payment.

After the customer *requires* the payment than the customer *waits for* funds.

After the customer *waits for* funds than the customer *generates* the payment.

After the customer *generates* the payment then the Maintenance-Unit *gets* the payment.

After the Maintenance-Unit *sends* the invoice then the Maintenance-Unit *handles* penalties.

After the Maintenance-Unit *handles* the penalty then the Maintenance-Unit *gets* the payment

While the Maintenance-Unit *handles* the penalty if the time-elapsed *is* 7 days then the Maintenance-Unit *gets* the payment.

Listing 6 – Refined Billing Process SBVR SE Ruleset

3.2.2 Considerations upon reconfiguration patterns and conditions

Changes applied to optimize the process model parallelizing the execution of "Prepare Invoice" and "Perform customer-acceptance-test" tasks correspond to a *Parallel Backward Shift* pattern.

Hence the admissibility conditions for process reconfigurability to be considered are the admissibility conditions for the above mentioned identified pattern.

Therefore the modification on a running instance can be admissible if the state of the target task we want to parallelize, i.e. the "Prepare Invoice" task, is in "initial", "ready" or "running" state.

3.3 Report on a reconfiguration case

In this section we analyze the reconfiguration patterns needed to modify the "Billing Process" describer in the previous sections. The initial version of the process is the "1.0", as shown in Figure 4.

Process name	Version	Description
Billing Process	1.0	

Figure 4 – Process initially deployed in the workflow engine

Business requirements (detailed in section 3.1) lead to the definition of a new version of the process model through the modification of its SBVR SE representation (listing 3 and 4). The new version is identified by the version number "1.1".

A new process model definition is automatically recognized when we try to deploy it in the workflow engine, comparing process name and version.

When a user tries to deploy a package in a workflow engine and a new version of a process model is identified, the reconfiguration admissibility analysis procedure is automatically started. The system understands that a previous version of the process definition is already deployed (see Figure 4) and then retrieves the XPDL file used to serialize the old and the new process model.

Comparing the two XPDL files, which are automatically obtained from the old (see listing 1 and 2) and the new (see listing 3 and 4) SBVR SE process model representations, we obtain that the reconfiguration pattern applied to the old process model in order to obtain the new version is Sequential Insertion. This is due to the insertion of the task "Perform acceptance-client-test". This situation is shown in Figure 5. In this case only one pattern has been identified, but in general, more than one pattern can be applied and recognized at the same time on a process model.

Pattern name	Task to check
Sequential insertion	Send invoice

Figure 5 – Reconfiguration patterns identified for the example when moving from process model version 1.0 to 1.1

Once reconfiguration patterns are identified, the system analyzes the state of execution for activities belonging to every running instance. A snippet of the status of execution is shown in Figure 6.

Task name	Instance name	Instance State
Get payment	Billing Process-1.0-Billing Process-1.0$1	finished
Send invoice	Billing Process-1.0-Billing Process-1.0$2	executing
Prepare invoice	Billing Process-1.0-Billing Process-1.0$3	executing
Prepare invoice	Billing Process-1.0-Billing Process-1.0$4	executing
Handle penality	Billing Process-1.0-Billing Process-1.0$5	executing
Prepare invoice	Billing Process-1.0-Billing Process-1.0$6	executing
Prepare invoice	Billing Process-1.0-Billing Process-1.0$7	executing
Prepare invoice	Billing Process-1.0-Billing Process-1.0$8	executing
Check invoice validity	Billing Process-1.0-Billing Process-1.0$9	executing
Notify penality	Billing Process-1.0-Billing Process-1.0$10	executing

Figure 6 – Activities state for the running process instances based on process model version 1.0

Then the software evaluates the admissibility of the migration for every instance. Results are shown in the Figure 7:

Instance name	Admissibility
Billing Process-1.0-Billing Process-1.0$2	No
Billing Process-1.0-Billing Process-1.0$3	Yes
Billing Process-1.0-Billing Process-1.0$4	Yes
Billing Process-1.0-Billing Process-1.0$5	No
Billing Process-1.0-Billing Process-1.0$6	Yes
Billing Process-1.0-Billing Process-1.0$7	Yes
Billing Process-1.0-Billing Process-1.0$8	Yes
Billing Process-1.0-Billing Process-1.0$9	No
Billing Process-1.0-Billing Process-1.0$10	No

Figure 7 – Results of running instances reconfigurability evaluation from version 1.0 to version 1.1

The result of the analysis is that instances whose names are:

- “Billing Process-1.0-Billing Process-1.0$3”;
- “Billing Process-1.0-Billing Process-1.0$4”;
- “Billing Process-1.0-Billing Process-1.0$6”;
- “Billing Process-1.0-Billing Process-1.0$7”;
- “Billing Process-1.0-Billing Process-1.0$8”;

can run accordingly to new process model.

Instead instances whose names are:

- “Billing Process-1.0-Billing Process-1.0$2”;
- “Billing Process-1.0-Billing Process-1.0$5”;
- “Billing Process-1.0-Billing Process-1.0$9”;
- “Billing Process-1.0-Billing Process-1.0$10”;

must follow the old process model definition because reconfiguration is considered not-admissible.

In particular in the following figures are shown the execution status of the tasks for two different instances, which names are “Billing Process-1.0-Billing Process-1.0$8”

(see Figure 8) and “Billing Process-1.0-Billing Process-1.0$2” (see Figure 9), to highlight how admissibility conditions are respectively verified and not-verified.

Name	Instance Name	State
Send Invoice	Billing Process-1.0-Billing Process-1.0$8	initial
Prepare Invoice	Billing Process-1.0-Billing Process-1.0$8	executing
Start	Billing Process-1.0-Billing Process-1.0$8	finished

Figure 8 - Tasks status for an admissible instance

Name	Instance Name	State
Send Invoice	Billing Process-1.0-Billing Process-1.0$2	executing
Prepare Invoice	Billing Process-1.0-Billing Process-1.0$2	finished
Start	Billing Process-1.0-Billing Process-1.0$2	finished

Figure 9 - Tasks status for a not-admissible instance

Immediately after this reconfiguration, a new analysis is needed to refine the process according to the modification described in section 3.2.

A new process model definition is specified in SBVR SE (see listing 3 and 5). Then the related XPDL file is automatically generated.

The system retrieves information about process model deployed in the workflow engine and performs the comparison between their version numbers.

Process name	Version	Description
Billing Process	1.1	

Figure 10 – Process already deployed on the workflow engine

Comparing the XPDL files related to the old (see listing 3 and 4) and the new (see listing 3 and 5) SBVR SE process model description, reconfiguration patterns

applied to the process model version 1.1 in order to obtain the new version 1.2 are identified. Results are shown in Figure 11.

Pattern name	Task to check
Parallel Backward Shift	Prepare invoice

Figure 11 – Reconfiguration patterns identified for the example when moving from process model version 1.1 to 1.2

As we previously specified (see 3.2.2) and as we notice in the above figure, parallelizing “Prepare Invoice” and “Perform customer-acceptance-test” tasks correspond to the application of the *Parallel Backward Shift* pattern.

Once patterns are identified, the system checks the status of every old-model-based running instance in order to evaluate the possibility of reconfiguration: admissibility conditions for patterns identified are analyzed. A snippet of the status of execution is shown in the Figure 12.

Task name	Instance name	Instance State
Get payment	Billing Process-1.1-Billing Process-1.1$1	finished
Notify penality	Billing Process-1.1-Billing Process-1.1$2	executing
Check invoice validity	Billing Process-1.1-Billing Process-1.1$3	executing
Prepare customer acceptance tests	Billing Process-1.1-Billing Process-1.1$4	executing
Handle penality	Billing Process-1.1-Billing Process-1.1$5	executing
Prepare invoice	Billing Process-1.1-Billing Process-1.1$6	executing
Prepare invoice	Billing Process-1.1-Billing Process-1.1$7	executing
Send invoice	Billing Process-1.1-Billing Process-1.1$8	executing
Prepare invoice	Billing Process-1.1-Billing Process-1.1$9	executing
Prepare invoice	Billing Process-1.1-Billing Process-1.1$10	executing

Figure 12 – Activities state for the running process instances based on process model version 1.1

The reconfiguration admissibility for a single instance depends on its own status. Finally results of this check are represented in Figure 13.

Instance name	Admissibility
Billing Process-1.1-Billing Process-1.1$2	No
Billing Process-1.1-Billing Process-1.1$3	No
Billing Process-1.1-Billing Process-1.1$4	No
Billing Process-1.1-Billing Process-1.1$5	No
Billing Process-1.1-Billing Process-1.1$6	Yes
Billing Process-1.1-Billing Process-1.1$7	Yes
Billing Process-1.1-Billing Process-1.1$8	No
Billing Process-1.1-Billing Process-1.1$9	Yes
Billing Process-1.1-Billing Process-1.1$10	No

Figure 13 – Results of running instances reconfigurability evaluation from version 1.1 to version 1.2

As we can also see in the above figure, results in this case shows how instances whose names are:

- “Billing Process-1.1-Billing Process-1.1$6”;
- “Billing Process-1.1-Billing Process-1.1$7”;
- “Billing Process-1.1-Billing Process-1.1$9”;

can be migrated to run accordingly to new process model without any loss of data. Instead instances whose names are:

- “Billing Process-1.1-Billing Process-1.1$2”;
- “Billing Process-1.1-Billing Process-1.1$3”;
- “Billing Process-1.1-Billing Process-1.1$4”;
- “Billing Process-1.1-Billing Process-1.1$5”;
- “Billing Process-1.1-Billing Process-1.1$8”;
- “Billing Process-1.1-Billing Process-1.1$10”;

must run accordingly to the old process model definition because reconfiguration is considered not-admissible.

The execution status of the tasks for two different instances are shown in the following, which names are “Billing Process-1.1-Billing Process-1.1$7” (see Figure

14) and “Billing Process-1.1-Billing Process-1.1$4” (see Figure 15), to highlight how admissibility conditions are respectively verified and not-verified.

Name	Instance Name	State
Prepare invoice	Billing Process-1.1-Billing Process-1.1$7	executing
Start	Billing Process-1.1-Billing Process-1.1$7	finished

Figure 14 - Tasks status for an admissible instance

Name	Instance Name	State
Perform customer acceptance tests	Billing Process-1.1-Billing Process-1.1$4	executing
Prepare invoice	Billing Process-1.1-Billing Process-1.1$4	finished
Start	Billing Process-1.1-Billing Process-1.1$4	finished

Figure 15 - Tasks status for a not-admissible instance

4 Conclusions

In this deliverable we wanted to show with a simple but real case what may be situations in a real business in which the reconfiguration is needed and how to manage this situation. First of all, reconfiguration can take place to meet with a change in business requirements. In a second step, the need to re-model a process arises to improve the performance of the process regarding execution time and resources exploitation. Then a reengineering action is started to refine the process.
In this example, which is taken from the real world, it was shown how to apply the tools developed to define a process model using a language (i.e. SBVR SE) more similar and close to the natural language used by business people. The tools made available allow business people to have control and to manage the process, allowing them to modify and adapt the process model in order to rapidly react to changes in business requirements typical of a dynamic environment, even allowing him to assess the applicability of the introduced modifications on process model in the various running instances of this process.

Although for migration of instances from one process model to another modified model it will take a further development of open source workflow engine solutions, the work done represents a fundamental step for the implementation of such solutions, since identification of the conditions under which such migration can occur without inconsistencies and data loss is the long standing problem of dynamic reconfiguration of the workflows. The solution proposed allows focusing, from now on, on the technical problems related with the implementation of a software solution to manage this functionality.
In the immediate future we will examine how developed tools will be integrated into OKS platforms, at this point in the implementation phase, and how they can be made available to the end user in a more organic way.

Bibliography

[1] Raimund Eder, Thomas Kurz, Thomas J. Heistracher, Victor Bayon, Mario Russo, and Antonella Filieri. **D2.1 - *Design of Software Generation Prototype***. OPAALS Project, October 2007.

[2] Raimund Eder, Thomas Kurz, Thomas J. Heistracher, Mario Russo, Antonella Filieri, Prabhakar TV, Hagen Peukert, Alexandros Marinos, Stan Hendryx. **D2.2 - *Automatic code structure and workflow generation from natural language models***. OPAALS Project, March 2008.

[3] Antonella Filieri, Antonio Margarito, Raimund Eder, Thomas Kurz. **D2.3 – *Extended vocabulary and rule set for an existing scenario*.** OPAALS Project, November 2008.

[4] Antonella Filieri, Antonio Margarito, **D2.4 – *Prototypical implementation of dynamic workflow reconfiguration.*** OPAALS Project, March 2009.

[5] WfMC. **XML Process Definition Language (XPDL 1.0)**, October 2002. http://www.wfmc.org/xpdl.html.

[6] WfMC. XML Process Definition Language (XPDL 2.1), March 2008. http://www.wfmc.org/xpdl.html.

[7] Nova Bonita Workflow Engine. http://wiki.bonita.ow2.org/xwiki/bin/view/Main/Nova

[8] OMG. Business Process Modeling Notation (BPMN 1.1), February 2008. http://www.bpmn.org/.

[9] OMG. Semantics of Business Vocabulary and Business Rules Specification (SBVR), January 2008. http://www.omg.org/spec/SBVR/1.0/PDF/.