Business Process Configuration Wizard and Consistency Checker for BPMN 2.0

Andreas Rogge-Solti, Matthias Kunze, Ahmed Awad, and Mathias Weske

Hasso Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
{andreas.rogge-solti, matthias.kunze, ahmed.awad,
matthias.weske}@hpi.uni-potsdam.de

Abstract. A rapidly changing environment, in terms of technology and market, forces companies to keep their business processes aligned with current and upcoming requirements. This is still a major issue in modern process oriented information systems, where improvements on process models require considerable effort to implement them in a technical infrastructure.

We address this problem by lifting technical details into BPMN 2.0 process models and present a configuration wizard for these process models in the open-source modeling tool Oryx. This wizard includes a consistency checking mechanism to automatically discover inconsistencies in the data dependencies of a process model. Immediate feedback after changes to the model eliminates a crucial source of errors when configuring or redesigning business process models, leading to more efficient process implementation.

Keywords: consistency checking, business process redesign, business process configuration, tool support.

1 Introduction

Within the last decade, business process management gained increasing impact on the way organizations conduct their businesses. Business process management is the key instrument to understanding and organizing the activities an organization undertakes to produce goods or deliver services. As businesses and their environment undergo continuous dynamics, the processes of an organization have to be steadily improved. However, adopting changes is still one of the biggest issues of process oriented information systems [12] and process management tools provide only limited support.

In this paper we present a configuration wizard that uses a model driven approach to leverage graphical process models, enrich them with technical details, and automatically generate process representations that can be readily enacted by a process engine. The wizard is built into the open-source modeling tool Oryx and includes a consistency checker for data flow in the process. The focus of this approach is on graphical models, since changes are most easily and consistently performed on a graphical representation. This enables an easy configuration of process models and quick redesign of existing process models.

T. Halpin et al. (Eds.): BPMDS 2011 and EMMSAD 2011, LNBIP 81, pp. 231–245, 2011.

The remainder of the paper is organized as follows. Section 2 introduces preliminary definitions. Section 3 elaborates on particular problems of the traditional approach to business process configuration. Section 4 presents conceptual ideas to solve these issues. In Section 5 we demonstrate a prototypical implementation of our approach. Section 6 compares our ideas with related work and finally, Section 7 concludes this paper and outlines future work.

2 Preliminaries

Let us first review the lifecycle of processes that are enacted with the help of information systems. Fig. 1 depicts the lifecycle's four most prominent phases, c.f. [1].

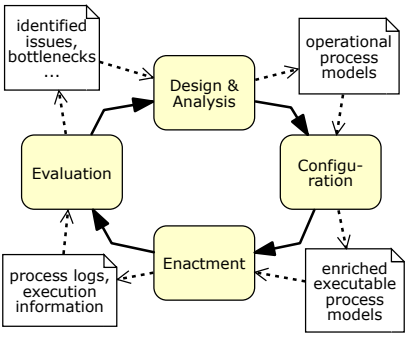

Fig. 1. BPM lifecycle

In the *design and analysis* phase, business processes are elicited and captured in graphical process models. These models need to be configured to be enacted on a process engine, which is done in the *configuration* phase. The resulting artifacts are executable process definitions. In the *enactment* phase, these process definitions are carried out by a process engine. Usually human interaction is embedded in the process and application services are called by the engine to integrate existing software systems. During *enactment*, process engines produce logs containing execution details of the process instances. The logs and runtime information can be analyzed in the *evaluation* phase. The cycle is re-iterated when the results of this phase are fed back into the analysis and design phase to improve the process.

Definition 1 (Process Model). *A process model P is a connected, directed graph (N, E) where $N = T \cup G \cup \{n_{in}, n_{out}\}$ is a finite set of nodes, with tasks T, gateways G, with $T \cap G = \emptyset$, and exactly one start and end event n_{in} and n_{out}, and $E \subseteq (N \setminus \{n_{out}\}) \times (N \setminus \{n_{in}\})$ is a set of edges connecting the nodes.*

In Definition 1, we capture process models as they are elicited in the *design and analysis* phase. They define activities and control flow structure. However, if enactment of such processes is desired, they need to be transformed into an executable format. This is done in the *configuration* phase. The business process execution language (BPEL) [3] is an industry standard that has clear execution semantics and can be executed by a process

engine, e.g., Apache ODE[1]. A common method is to translate graphical process models, e.g., defined as EPC [7] or BPMN [13] diagrams, into an executable representation, e.g., BPEL, cf. [21,14], and extend the generated code skeleton with execution details, as we describe next.

User tasks enable users to interact with the process and thus need a user interface. Usually these user interfaces are forms to display or enter data. Some activities in the process can be executed automatically, for instance, through running a script, calling an application or a Web service. This is realized through a service task that needs configuration to execute the corresponding logic. Both task types, user task and service task, may access data objects during the enactment of the process. Further, process control flow may diverge on databased exclusive gateways (XOR splits), depending on values of data objects of process instances. The technical engineer needs to assign the proper expressions at the XOR splits in the process model.

Definition 2 (Data Access). *Let D be the set of data objects within a process model P. A node can either read (r), optionally read (or), write (w), optionally write (ow), or not process (n) a data object. The* data access matrix *of nodes to data objects is a function: access* : $N \times D \rightarrow \{or, r, ow, w, n, (or, ow), (or, w), (r, ow), (r, w)\}$

Roles need to be configured for each user task defined in the process. If the concept of pools and lanes is used for role modeling, it is much easier for the technical engineer to configure the roles on the lane level. This solution greatly speeds up configuration of business processes with many user tasks. Likewise service tasks need to be configured to call a specific service.

Definition 3 (Assignment). *Let R be a set of roles that map to organizational functions and S a set of application services of that organization. The assignment of task $t \in T$ to a role $r \in R$ or a service $s \in S$ is a function: assign* : $T \rightarrow R \cup S$

The above steps yield a configured process model that can be executed on a process engine.

Definition 4 (Configured Process Model). *A configured process model $P_{conf} = (N, E, D, R, access, assign)$ is a process model, where each user task $T_{User} \subseteq T$ is assigned to a role and each service task $T_{Service} \subseteq T$ is assigned to a service, where $T_{User} \cup T_{Service} = T \bigwedge T_{User} \cap T_{Service} = \emptyset$. $G_{XOR} \subseteq G$ is the set of data-based exclusive gateways that define for each outgoing edge an expression based on data objects, which is captured by the access function*

3 Hardships in the Configuration of Processes

In traditional approaches, as we observe in many modern business process management systems, both model types from Definition 1 and 4 are decoupled. Business experts graphically design process models in the *design and analysis* phase and hand the results to the IT-department, where these models are transformed into executable formats (see above) and then need to be configured manually.

[1] Apache ODE – `http://ode.apache.org`

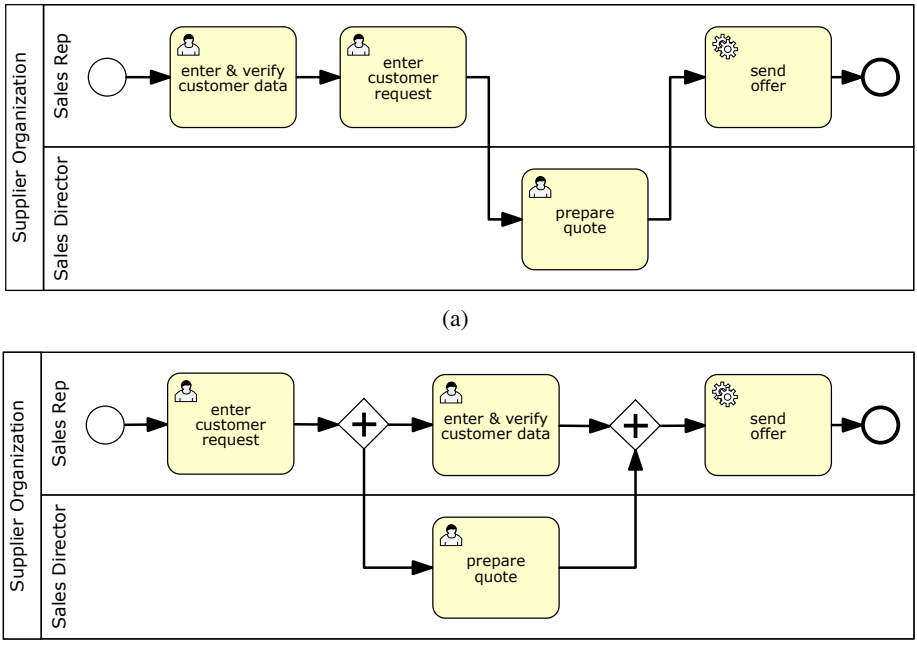

Fig. 2. Example process before (a) and after (b) redesign

Besides the error-proneness of this approach, this method has a major problem dealing with changes in the process model. If changes are made in the process model, it is a cumbersome task to keep the configured, executable process synchronized with the model; if changes are required on the technical model, they would need to be incorporated into the graphical model, which rarely happens, because again, these are manual tasks. The process models serve mainly for documentation purpose and quickly get out of sync with the reality.

Another hardship is to keep track of all data dependencies in a process model that need to be considered when configuring or redesigning a process model. While methods to ensure control flow consistency of process models where introduced more than a decade ago [2], dependencies between the data object access of nodes still expose a high risk of inconsistencies, and may require great effort to disclose.

Consider the example depicted in Fig. 2. Note that it is not obvious, which data dependencies exist in the model, i.e., upon changing the model with the intention of improving the process, one may unintendedly destroy the data flow consistency.

The example (Fig. 2(a)) shows a simplified process of a manufacturer company, where a sales representative stays in contact with a customer, who requested an offer. The sales representative enters the data of the customer, e.g., name and address, as well as the request of supplied goods. However, this person may not be authorized to create a quote, this is the task of the sales director for which he needs only information about the customer's request. After all information is set, the sales representative can send the offer to the customer.

Based on the assumption that the configuration of the process model in Fig. 2(a) is complete and error-free, to improve the runtime of this process, the tasks to enter the customer's data and request, and to create the quote could be conducted in parallel, as they are user tasks that may take a considerable amount of time. However, the task *prepare quote* requires the customer's request data, i.e., they have causal data dependencies. Any attempt to model them in parallel would lead to a missing data inconsistency. Since the data dependencies of this process model are hidden in technical details and not obvious, such violations wouldn't be disclosed until the process becomes implemented.

A better solution is depicted in Fig. 2(b) and could only be proposed if one has detailed insight into the data that is read and written by each task: Entering the customers data does not require the customer's request, nor does it require the prepared quote. From that, it follows that the sales representative could *enter the customer request* first, and while they *enter & verify the customer's data* the sales director can *prepare the quote*. Both concurrent paths modify distinct data objects and are synchronized before the offer is sent to the customer. Thus, no data dependencies are violated and no inconsistencies occur and the process is correctly redesigned.

In the next sections we present a set of redesign patterns that guide the improvement of business processes and the fundamentals of checking data flow consistency. By means of a prototypical implementation we show how we addressed the hardships identified above, i.e., manual configuration of technical details, poor support of data flow consistency checking, and the lack of roundtrip support in the business process life cycle.

4 Alleviating the Hardships

In the configuration phase of the business process lifecycle, tool support for changing business processes is most important. To avoid the above mentioned synchronization issues between process models and technical implementations, we merge them, i.e., we add technical details to the operational graphical models and generate executable process definitions of these models. While this concept is not new and quite common in model driven engineering [10], applied to process modeling it has to be altered in a domain specific way.

To reduce possible errors in the configuration phase, we designed a wizard guiding the technical engineer through these configuration steps. The first requirement to the wizard is that instead of typing names of resource assignments in the process models, the technical engineer should be able to select the resources from a list of available ones. Besides making sure, that only existing resources can be assigned, it is important for a process model to be executed correctly, that all data-dependencies in the process model are satisfied. We go into the inner mechanics of the data-dependency check in Section 4.2. But first, we discuss the impact of the consistency checker on business process redesign.

4.1 Supported Redesign Patterns

Support for process model changes, e.g., redesign, is not sufficiently available in current process aware information systems [12]. To better understand, what can be done

to support these changes, it is necessary to know what types of changes should be supported. Addressing this question, Reijers et al. identified 29 general redesign patterns for improving business processes [15]. Table 1 lists the patterns, that are reflected on the process model layer. These redesign patterns can only be applied correctly under certain preconditions regarding data flow. In the following, we define these preconditions and provide examples of how we support these redesign patterns.

Table 1. List of supported process model redesign patterns [15]

Redesign pattern	Support
operational patterns	
Order types	`possible`
Task elimination	`yes`
Triage	`yes`
Task-composition	`possible`
behavior patterns	
Resequencing	`yes`
Parallelism	`yes`
information patterns	
Control addition	`yes`
technology patterns	
Task automation	`yes`

Order types describes the problem of treating all instances in a process equally, even though some instances do not require all process steps.

pre: none

example: The subprocess "Packaging" of an order process is split into two different variants. One for single order positions and another one for multiple positions. The subprocess for single positions does not need the extra task "Choose delivery options", where the customer can choose whether to wait for all positions to be sent in one package.

requirements to the wizard: 1) The wizard could be extended to find violations of data consistency in process hierarchies. However, the link between parent and child model must be bidirectional for consistency checks, when changing a child model, that is referenced in many parents. 2) Dealing with variants can be implemented similarly, the only difference is that the link between the "parent" and "child" is horizontal instead of vertical.

Task elimination.

pre: A task $t \in T$ to be deleted may only write data $d \in D$ that is not read later in the process, or that was already available in the process. This includes data that contributes to the output of the process.

example: In a production process, there is a task "Check quality" at the end. It can be eliminated to save cost, since it does not produce data. The external behavior of the production process is still the same, however the non-functional property "quality" may have been reduced by this pattern.

support: The configuration wizard immediately gives feedback, if data produced by an eliminated task is required somewhere in the process.

Triage refers to (a) splitting a general task into several alternatives based on the instance data, to better utilize resources and make use of specialization. Also the contrary is possible, i.e., to (b) combine alternative tasks to a general task.

pre: (a) none (to split a task into alternatives)
(b) none (to join alternatives)

example: The task "Check order" can be split into "Check high volume order" and "Check low volume order". Assigning these tasks to different people in the organization may improve performance of the process through specialization.

support: The wizard supports copy-&-paste for tasks, allowing to split a task easily. After wiring the control flow and adding the data based routing gateway, the technical engineer needs to define the routing conditions of the gateway. The consistency checker immediately warns, if the routing conditions are invalid.

Task composition (a) combines small task into composite tasks or (b) divides large tasks into workable smaller tasks.

pre: (a) none (to split a task into several single steps)
(b) none (to merge tasks)

example: In a registration process, the two sequential tasks "Insert customer details" and "choose password" can be merged. The two respective input forms will be merged too and be assigned to the merged task.

requirements to the wizard: In order to properly support this redesign pattern, we plan to have a graphical editing of user forms used in the configuration wizard. To support task splitting, the user should be able to mark specific input fields of a selected task's form. The system then can split the form into two parts. Where one contains the marked input fields, and the other the remaining ones. The task itself will be duplicated and added in the control flow right behind the first task. Task merging would reverse this procedure, i.e., two sequential user tasks can be merged into one by combining the respective forms of each and assigning it to the resulting task.

Resequencing.

pre: Task *a* and *b* are in sequence, i.e., *b* is done after *a*. Task *b* may not read data that is produced only by *a* or later tasks in the process.

example: In the motivating example in Fig. 2(b), the tasks "enter customer request" and "enter & verify customer data" are resequenced, because they do not share any data dependency. However, they are conducted by the same person, which motivates to keep them in sequence.

support: Process modelers can switch tasks in sequence and get notified, whether there arise conflicts by doing so.

Parallelism.

pre: a) Additionally to the pre-condition of *Resequencing*, both tasks must not write the same data element. (to make sequential tasks parallel)
b) none (to sequentialize two parallel tasks)

example: In the motivating example in Fig. 2(b), the tasks "enter & verify customer data" and "prepare quote" are made parallel, because they do not access data produced by one another nor do they write to the same data object.
support: After rearranging the control flow in the process, the process modeler gets immediate feedback, if the data-dependencies still hold, and a successful execution of the process is possible.

Control addition means adding an extra quality control task to check the results of former tasks.
pre: none
example: After the "Approve credit" task performed by a clerk in a credit agency's core business process, a new "Verify credit approval" task is added. This new task will be performed by the controlling department, which can review the decision of the clerk and overrule it in case.
support: In the process modeler used by the wizard the task to be controlled can be copy-&-pasted. The new copy of the task needs to be assigned to another functional role that has the authorization to perform the quality control. Renaming the control task and connecting the nodes completes this pattern.

Task automation.
pre: The user task $u \in T_{user}$ to be automated should have an equivalent service implementation s configured in the system. Service s has to write at least the subset of written data from u, that is used in the process later on.
example: The "Send invoice" task of an order process is to be automated. This is done by manually sending an email, and will be done by a service afterwards.
support: Once the service is implemented and configured, the technical engineer has to change the type of the task to be automated from *user*- to *service*-task and select the existing implementation from the list of configured services.

4.2 Consistency Checking

As indicated earlier, checking consistency of data dependencies and making sure that no data anomalies will be manifested through execution is of utmost importance. For instance, if some task within a process reads a data object that is not yet initialized, this might lead to deadlocks in the process execution.

Data anomalies can be categorized into three basic categories: *missing data*, *redundant data*, *conflicting data* [17]. As indicated above, a missing data anomaly occurs in general when there is a chance for some task to be executed where it reads a data object that was never written (initialized) before. Redundant data anomaly occurs when a data object is written by some task but never read by any subsequent task. Finally conflicting data anomaly occurs when a data object is written by two or more tasks concurrently.

Sun et al. in [17] have given a comprehensive description of such anomalies and how to detect them, due to space limitations, we will discuss a core subset of such anomalies. Basically, we need two pieces of information to detect such anomalies: The read/write access taken by each node t against each data object d and the execution ordering, i.e., behavioral relationships, among nodes within the process model. In their work [17], the

Table 2. Data anomalies

Anomaly	Description
Missing data warning type 1	A task b reads an object that is optionally written by a preceding task a
Missing data warning type 2	A task b optionally reads an object that was never written by any preceding task
Missing data error	A task b reads an object that was never written by any preceding task
Redundant data warning	A data object is written by a task a but it was never read afterwards
Conflicting data warning	Two concurrent tasks a,b write a data object where at least one of them optionally writes the data object
Conflicting data error	Two concurrent tasks a,b write a data object

authors did not specify an exact algorithm to build the behavioral relationship among tasks. Therefore, we depend on the notion of behavioral profiles [20] to obtain the behavioral relationship among tasks within a process. A behavioral profile identifies the behavioral relationship between any pair of nodes within the model. This relationship is one of four values: (1) strict order $\rightsquigarrow$, (2) concurrent $\|$, (3) exclusive # or (4) inverse order $\leftsquigarrow$. Moreover, for each task, we need to know whether it is optional.

Definition 5 (Behavioral Profile). *Let N be the set of nodes within a business process model. The* behavioral profile *of a business process model is a function $bhp : N \times N \rightarrow \{\rightsquigarrow, \leftsquigarrow, \|, \#\}$ that assigns a behavioral property, strict order, inverse order, parallel, or exclusive, between each pair of nodes within the business process model.*

If two tasks a, b appear in strict order, $bhp(a,b) = \rightsquigarrow$, then task a executes before task b. Similarly, if two tasks are concurrent then they can be executed in any order. Exclusiveness means that at most one of the two tasks can execute within a process instance. A node $t \in N$ is *optional*, noted as $opt(t) = true$, if there is at least one other node $s \in N$ where $s \neq t$ and $bhp(s,t) = \#$. As the name indicates, the inverse order relation is the inverse of the strict order relation, $bhp(a,b) = \rightsquigarrow \Rightarrow bhp(b,a) = \leftsquigarrow$ and is defined for readability reasons only.

The other input to check consistency of data dependencies and lack of anomalies is the read/write relation among tasks and data objects. In Definition 2, it was shown that reading/writing a data object by a node can be optional. This, in turn, allows to generate finer grained levels of checks, e.g., warnings rather than errors. For instance, if a task b reads a data object d and another task a, where $bhp(a,b) = \rightsquigarrow$, optionally writes d then, the consistency checker will warn for a possible missing data anomaly. That is because, at execution time of b the data object d *might* not have been written by task a. On the other hand, if there was no task that writes d at all before b, an error is generated. Table 2 informally describes the different anomalies that can be identified with our approach. Also, it is possible to give hints to the user to help resolve such anomalies. For instance, for a missing data error, it could be possible that the task a that first writes a data object read by another task b executes after b, known as *late initialization* [17].

Based on definitions 5 and 4, we can define the different types of data anomalies: missing data, redundant data and conflicting data.

Definition 6 (Missing data Anomaly). *Let N be the set of nodes and D be the set of data objects within a business process model P_{conf} respectively. P_{conf} suffers from a* missing data anomaly *iff:*
$\exists t \in N\, \exists d \in D\, \nexists s \in N : \{r\} \in access(t,d) \wedge bhp(s,t) = \rightsquigarrow \wedge (\{w\} \in access(s,d) \vee \{ow\} \in access(s,d))$ *-Error.*
$\exists t \in N\, \exists d \in D\, \exists s \in N : \{r\} \in access(t,d) \wedge bhp(s,t) = \rightsquigarrow \wedge (opt(s) = true \vee \{ow\} \in access(s,d))$ *-Warning type 1.*
$\exists t \in N\, \exists d \in D\, \nexists s \in N : \{or\} \in access(t,d) \wedge bhp(s,t) = \rightsquigarrow \wedge (\{w\} \in access(s,d) \vee \{ow\} \in access(s,d))$ *-Warning type 2.*

Definition 7 (Redundant data Anomaly). *Let N be the set of nodes and D be the set of data objects within a business process model P_{conf} respectively. P_{conf} suffers from a* redundant data anomaly *iff:*
$\exists t \in N\, \exists d \in D\, \nexists s \in N : \{w\} \in access(t,d) \wedge bhp(t,s) = \rightsquigarrow \wedge (\{r\} \in access(s,d) \vee \{or\} \in access(s,d))$.

Definition 8 (Conflicting data Anomaly). *Let N be the set of nodes and D be the set of data objects within a business process model P_{conf} respectively. P_{conf} suffers from a* conflicting data anomaly *iff:*
$\exists t,s \in N\, \exists d \in D : \{w\} \in access(t,d) \wedge bhp(s,t) = \| \wedge \{w\} \in access(s,d)$ *-Error.*
$\exists t,s \in N\, \exists d \in D : \{w\} \in access(t,d) \wedge bhp(s,t) = \| \wedge (opt(s) = true \vee \{ow\} \in access(s,d))$ *-Warning .*

5 Prototypical Implementation

To validate the concepts discussed above, we implemented a tool that supports the aforementioned process model redesigns. It does not automatically identify deficiencies of a process model and apply the adequate pattern to improve it, but rather supports process experts, e.g., process engineers, to change the structure of the model according to these patterns and verify whether the improved model's data flow remains consistent.

This tool is based on Oryx[2]—an extensible process modeling platform that enables users to easily and efficiently create process models on the Web and allows developers to extend the tool's modeling capabilities. New modeling languages or extensions thereof can be added as stencil sets, new functionality can be made available to users through an elaborate plugin infrastructure [9]. Fig. 3 shows the example process model before redesign in Oryx.

As process definition language we resorted to a subset of BPMN 2.0 that suffices to capture most of today's process models, yet remains simple enough to be comprehended by the majority of users [11]. This subset allows to define tasks, to be conducted through software services or humans, control flow splits, either through exclusive choice or parallel branches, as well as pools and swim lanes to group activities according to the organizational role that conducts these tasks.

[2] The Oryx Research Project – `http://oryx-project.org/research`

To lift technical aspects of a process model implementation into the graphical model, we extended the BPMN 2.0 stencil set with properties that capture the configuration of roles, user tasks, system tasks, and XOR splits. Further, we implemented a wizard plugin that assists the process engineer to configure the process in three steps.

1. **Assign roles.** This is done through accessing a directory service, e.g., LDAP, and requesting the roles (user groups) of an organization. The technical engineer can choose for each lane, which group is responsible to conduct the tasks contained in the swim lane, cf. Definition 3.
2. **Configure tasks and control flow.** In this step, the main configuration takes place. Data forms are assigned to user tasks, services are registered with service tasks, and condition expressions of an XOR split are assigned. Data forms and services have been set up before, i.e., they are stored in a repository along with metadata that specifies input data, optional input data, and output data. By that metadata, the data consistency checker can derive, which data is read and which is written by the respective activity, cf. Definition 2.

   The wizard offers a configuration interface for each activity, where the engineer can select the data form or service, respectively. This is shown in Fig. 3. XOR splits are configured through writing expressions that use variables, or specifying at most

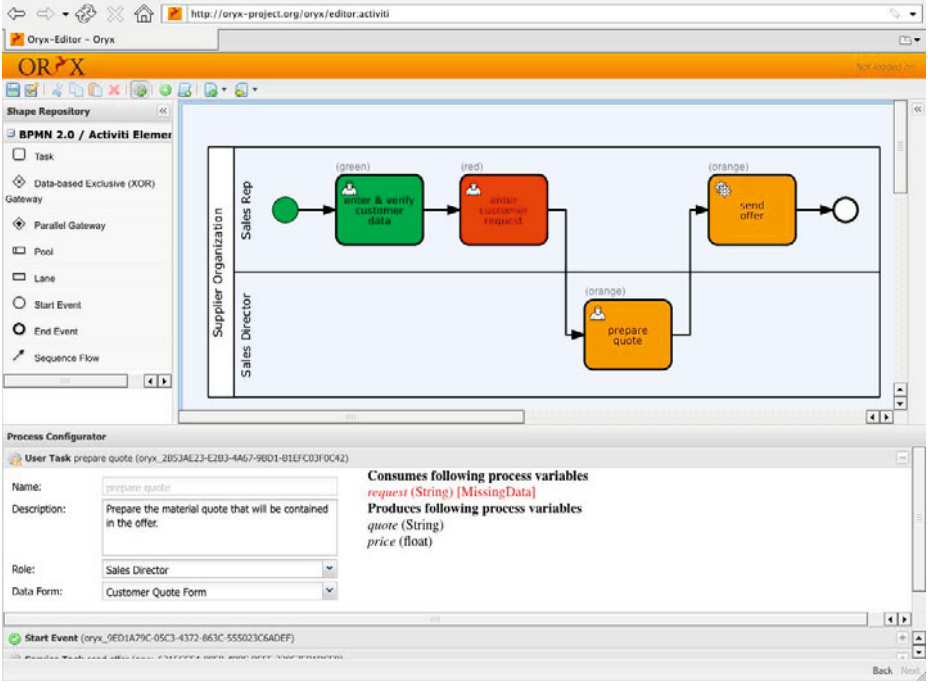

Fig. 3. Screenshot of the Configuration Wizard with an example business process model. The configuration wizard shows the user task “prepare quote”.

one edge as the default edge chosen when no other condition evaluates to true, cf. Definition 4

3. **Deploy process.** We implemented a deployment service, that transforms the process from the stencil set model into a readily executable format and installs it on a running instance of the open source BPM platform Activiti[3].

In the second step *Configure tasks and control flow*, every change triggers a (non-blocking) consistency check: Data dependencies are computed from data access and control flow, cf. Section 4.2, by deriving the behavioral profile for the nodes.

Any violations are displayed in the process model in the following way. If an activity or XOR split has not been configured sufficiently yet, it is highlighted in red. If it has been configured and no violations have been detected, it is highlighted green. If a task or XOR split has been configured, but consistency violations have been detected, the corresponding element is highlighted orange, and in the configuration interface the variables that caused the violations are shown in red along with the type of violation. In Fig. 3, this is the case for the tasks *prepare quote* and *send offer*, because the task *enter customer request* has not been configured yet, and thus the data object *request* is not available. As soon as this task is configured correctly, the data flow within the process will be consistent. Then, the process can be deployed to an engine.

6 Related Work

The first problem in configuring business process models is how to deal with the gap between the graphical models and the implemented versions interpreted by a process engine. As this problem is crucial in the popular research field of *model driven engineering*, there exist many conceptual solutions already. The most direct way to bridge the gap between models and implementation is to make models executable, by adding the necessary details [10]. We use the very same concept and apply it to configuring BPMN 2.0 models with necessary execution artifacts. Besides this method, also other techniques such as round-trip engineering were applied to process models [22]. This approach establishes associations between business workflow models and source code. These can be used, when it is not possible to integrate all information into one model.

The second problem is the efficient support for changes in graph-based models. In this area two simple refactorings for UML activity diagrams were introduced in [4], i.e., making actions parallel and sequentializing parallel actions. More recently 11 concrete business process model refactorings were defined in the context of large process repositories [19]. Tool support is still missing though, and these particular refactorings are mainly applicable only to process trees or process models with variants. For graph-based models, even automatic refactoring methods were proposed in [18]. These refactorings rearrange control-flow in the model, such that they get well-structured, i.e., can be split into hierarchical singe-entry single-exit regions.

Similarly to our continuous consistency checking for data dependencies, the authors in [8] present continous validation while modeling, but restrict the checks to structural properties, i.e., soundness of process models.

[3] Activiti BPM Platform – `http://www.activiti.org/`

Related to the *data dependency check* we presented in this paper is [16], where the authors identified a set of data anomalies as: redundant data, lost data, missing data, mismatched data, inconsistent data, misdirected data, and insufficient data. However, the paper just signaled these types of anomalies without an approach to detect them. A refinement of the aforementioned set of data anomalies was given in [17]. Although the authors in [17] provide algorithms to check such data anomalies, they do not discuss how to build the behavioral relationship within a model. In comparison to our work, we depend on the notion of behavioral profiles [20], which can be computed efficiently, to get the behavioral relationships among tasks within a process model. Then, we build a CRUD matrix to identify relationships among tasks and data items. Moreover, we provide a finer grained level of feedback. That is, we distinguish between warnings and errors. Recently, the authors in [6] presented an algorithm for efficient detection of data anomalies in business processes. They also provide detailed feedback and produce errors and warnings. However, they require the models to be structured to be able to analyse them.

Van Hee et al. [5] present a case study on how consistency between models of different aspects of a system can be achieved. They model object life cycles derived from CRUD matrices as workflow nets and later synchronized with the control flow. Their approach, however, does not present strategies how inconsistencies between data flow and control flow can be removed. Compared to our approach, we do not require the knowledge of object life cycles before hand. Rather, we extract it from the model under investigation.

7 Conclusion

Supporting changes in business processes is a major requirement in industry, that is not yet sufficiently solved. In this paper, we addressed this problem by augmenting the configuration and (re-)design phases of the BPM lifecycle with a configuration wizard relying on a model driven approach. The configuration wizard is integrated in the open-source modeling platform Oryx and builds on BPMN 2.0 process models. Continuous consistency checks ensure that data-related modeling errors are detected immediately, saving a lot of effort to detect potential errors manually. Thus, changes in the process models are facilitated and encouraged.

We further identified seven redesign patterns in [15] and explained how they are supported by the wizard. We introduced a fine grained definition of data-anomalies that can occur in business processes and explained how consistency checks can detect them. Finally, we compared our solution with existing work in these areas.

Future work includes providing support for the two redesign patterns we do not support yet, i.e., *Order types* and *Task composition*. Also, we want to integrate a recommender system based on the consistency checking results. Cases of *late instanciation*, mentioned in Section 4.2, that refer to read access to a process variable, that is written later in the process can be detected and pointed out to the technical engineer. Other suggestions, along the lines of *"task A reads data d, which is missing in the process, but provided by form f (or service s)"* can also be provided to the technical engineer configuring the process.

As we mentioned in Section 5, the current approach is limited to a subset of the most commonly used BPMN 2.0 modeling constructs, cf. [11]. As engines will support more advanced process constructs, such as event-based gateways, timer events, etc, we shall extend the wizard and data consistency checker with them.

References

1. van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M.: Business process management: A survey. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 1–12. Springer, Heidelberg (2003)
2. van der Aalst, W.M.P.: Workflow verification: Finding control-flow errors using petri-net-based techniques. In: van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 161–183. Springer, Heidelberg (2000)
3. Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guízar, A., Kartha, N., et al.: Web Services Business Process Execution Language Version 2.0. OASIS (2007)
4. Boger, M., Sturm, T., Fragemann, P.: Refactoring browser for UML. In: Liu, Y., Awasthi, P., Unland, R. (eds.) NODe 2002. LNCS, vol. 2591, pp. 366–377. Springer, Heidelberg (2003)
5. Hee, K.M.v., Sidorova, N., Somers, L.J., Voorhoeve, M.: Consistency in model integration. Data Knowl. Eng. 56(1), 4–22 (2006)
6. Heinze, T.S., Amme, W., Moser, S.: Effiziente Abschätzung von Datenflussfehlern in strukturierten Geschäftsprozessen. In: CEUR Workshop Proc., vol. 705 (2011)
7. Keller, G., Nüttgens, M., Scheer, A.: Semantische Prozeßmodellierung auf der Grundlage "Ereignisgesteuerter Prozeßketten (EPK)". Inst. für Wirtschaftsinformatik (1992)
8. Kühne, S., Kern, H., Gruhn, V., Laue, R.: Business Process Modelling with Continuous Validation. In: Workshops of BPM. LNBIP, vol. 17, Part 3. Springer, Heidelberg (2008)
9. Decker, G., Overdick, H., Weske, M.: Oryx – sharing conceptual models on the web. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 536–537. Springer, Heidelberg (2008)
10. Mellor, S., Balcer, M.: Executable UML: A foundation for model-driven architectures. Addison-Wesley, Reading (2002)
11. zur Muehlen, M., Recker, J.C.: How much language is enough? Theoretical and practical use of the business process modeling notation. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 465–479. Springer, Heidelberg (2008)
12. Mutschler, B., Reichert, M., Bumiller, J.: Unleashing the Effectiveness of Process-Oriented Information Systems: Problem Analysis, Critical Success Factors, and Implications. IEEE Trans. Syst., Man, Cybern. 38(3), 280–291 (2008)
13. OMG: Business Process Model and Notation (BPMN) 2.0 Specification (January 2011)
14. Ouvans, C., Dumas, M., ter Hofstede, A.H.M., van der Aalst, W.M.P.: From BPMN process models to BPEL web services. In: ICWS 2006, pp. 285–292. IEEE, Los Alamitos (2006)
15. Reijers, H., Liman Mansar, S.: Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics. Omega 33(4), 283–306 (2005)
16. Sadiq, S., Orlowska, M., Sadiq, W., Foulger, C.: Data flow and validation in workflow modeling. In: ADC, pp. 207–214 (2004)
17. Sun, S.X., Zhao, J.L., Nunamaker, J.F., Sheng, O.R.L.: Formulating the data-flow perspective for business process management. Info. Sys. Research 17(4), 374–391 (2006)
18. Vanhatalo, J., Völzer, H., Leymann, F., Moser, S.: Automatic workflow graph refactoring and completion. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 100–115. Springer, Heidelberg (2008)

19. Weber, B., Reichert, M.: Refactoring process models in large process repositories. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 124–139. Springer, Heidelberg (2008)
20. Weidlich, M., Polyvyanyy, A., Mendling, J., Weske, M.: Efficient computation of causal behavioural profiles using structural decomposition. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 63–83. Springer, Heidelberg (2010)
21. Ziemann, J., Mendling, J.: EPC-based modelling of BPEL processes: a pragmatic transformation approach. In: MITIP 2005. Citeseer (2005)
22. Zou, Y., Lau, T., Kontogiannis, K., Tong, T., McKegney, R.: Model-driven business process recovery. In: WCRE 2004, pp. 224–233. IEEE, Los Alamitos (2004)