BPModelMasher: Manage Your Process Variants Effectively

Sherif Sakr[1], Emilian Pascalau[2], Ahmed Awad[2], Mathias Weske[2]

[1] NICTA and University of New South Wales, Australia
ssakr@cse.unsw.edu.au
[2] Hasso-Plattner-Institute, University of Potsdam, Germany
{emilian.pascalau, ahmed.awad, mathias.weske}@hpi.uni-potsdam.de

Abstract. Nowadays, modern organizations build large repositories of process models to describe and document their daily business operations. One reason for the large number of process models is the need to adapt with differnt business contexts, i.e. process variants. Automated maintenance of the consistency between process variants is an important goal that saves the time and efforts of process modelers. We present a query-based approach to maintain consistency among process variants called *BPModelMasher*. In particular, we maintain the link between the variant process models by process model views. These views are defined using, BPMN-Q, a visual query language for process models. Dynamic evaluation for the defined queries of the process views guarantee that the process modeler is able to get up-to-date and consistent status of the process model. In addition, our view-based approach allows building multiple configurations for a holistic view of related variants of the same process model. The conceptual results are illustrated with a real-world sample process on customer service from eBay.

1 Introduction

Business process models play an effective role in providing a better understanding of the business and facilitating communication between business analysts and IT experts. The intensive use of business process models has its flip side. In practice, business processes do not exist only under a single version which covers all the issues of the whole market. Instead, many variants of a process may exist within the same enterprise in order to deal with different business situations such as: targeting different customer types, relying on particular IT systems or complying with specific national regulations. Therefore, we can establish an analogy between process variants on the one hand and object oriented inheritance on the other hand. A *process variant* is like a child class which extends or overrides the behavior of the parent process. Usually, these variants are maintained manually. A direct result of this manual maintenance is the risk of inconsistency. This inconsistency appears when a parent process's behavior is updated without updating the child's behavior accordingly [7,8]. Nevertheless, multinational companies, e.g, eBay, have to keep many variants of business processes in order to deal with the previously mentioned different business situations. Currently, a *Save-as* approach is followed to create such variants which leads to losing the link between related models and makes it possible to have many redundant models and inconsistent situations.

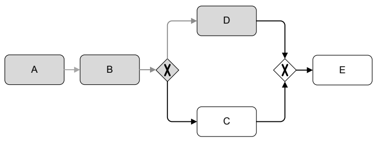
The goal of this demonstration is to present a novel framework for simplifying designing and managing business process model variants called *BPModelMasher*. The framework takes advantage of process model repositories during the design time in order to facilitate the reuse of *process model fragments*. By means of *partial process models*, the user can model new variant processes efficiently. Each partial process model is basically using two sets of notations: 1) Common process modeling constructs (e.g., tasks, control routing, control flow edges) which are used to imperatively describe the new functionality provided by the process model. Elements of a partial process model created using this notation are called the *concrete* parts of the model. 2) A set of notation which is used to create views on the inherited behavior from the parent process. To create these views, we use BPMN-Q [1], a visual language to query business process models. Query elements of the partial process model are called the *variable* parts. Using partial process models, we replace the manual *save-as* style of processes to create variants with an approach that keeps the link between *child* and *parent* processes by means of defining process views (queries).

Our framework is built on top of the open modeling platform and repository Oryx [5], and the BPMN-Q query language [1,10] as a means to access and retrieve process components from the process model repository. In particular, we summarize the main strengths of our demonstrated system as follows: 1) It reduces the time and effort of the business process modeling task [10]. 2) The enabled reuse and view-definition facilities on the process fragment level improve the quality and maturity of the newly developed variant process models and relaxes the learning curve, particularly for novices in a business domain [2]. 3) It facilitates the integration of different *process views* for a set of underlying low-level process models and automatically maintains the consistency among them [7]. This can be seen as supporting multiple inheritance among process variants.

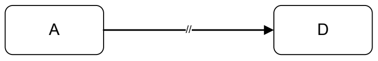
2 Real-World Use Case of View-Based Management for Process Variants

Currently, there are a number of graph-based business process modeling languages (e.g., BPMN and EPC). Despite their variance in expressiveness and modeling notation, they all share the common concepts of tasks, events, gateways (or routing nodes), artifacts, and resources, as well as relations between them, such as control flow. To define views over parent processes, we use the BPMN-Q notation [1,10]. The language supports, by means of visual notations, querying all the control and artifact concepts of business process models. Moreover, it introduces a set of new *abstraction* concepts that are useful for different querying scenarios. The structural matching of a query against a process results in a process subgraph (process fragment). In case that the subgraph is empty, we know that there is no match. Otherwise, the subgraph represents the matching part of the process to the query. Figure 1(a) illustrates a sample process model definition using the BPMN notations, Figure 1(b) illustrates a sample definition of a process model view using the BPMN-Q notations and that nodes and edges highlighted in grey in Figure 1(a) illustrate the matching part of the process model.

With more than 90 million active users globally, eBay is the world's largest online marketplace. eBay connects individual buyers and sellers, as well as small businesses

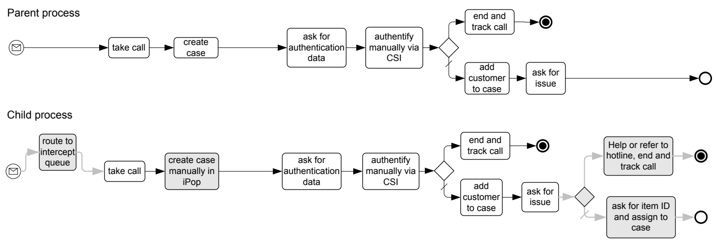


(a) Sample business process model



(b) A BPMN-Q query

Fig. 1. Example matching process between a sample process model and a BPMN-Q query


Parent process
take call
create case
ask for authentication data
authentify manually via CSI
end and track call
add customer to case
ask for issue
Child process
route to intercept queue
take call
create case manually in iPop
ask for authentication data
authentify manually via CSI
end and track call
add customer to case
ask for issue
Help or refer to hotline, end and track call
ask for item ID and assign to case
(a) An Example of Parent Process Model vs. Child Process Model

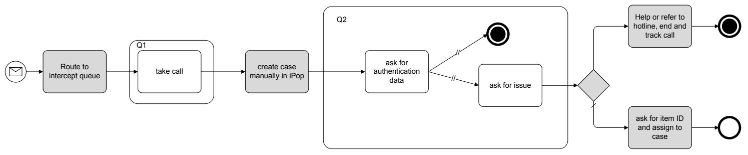

Route to intercept queue
Q1
take call
create case manually in iPop
Q2
ask for authentication data
ask for issue
Help or refer to hotline, end and track call
ask for item ID and assign to case
(b) A Partial Process Model to Express Inheritance Among Process Variants

Fig. 2. View-based management of process variants

in 38 markets using 16 languages [3]. Therefore, eBay has huge repositories of business processes. However, many of these processes are variants of other processes. The variability is imposed on a vertical axis (represented by different departments within the organization) and on a horizontal axis (emphasized by different business elements and/or business aspects, i.e., regulations, IT infrastructure, customer types, countries, payment methods). In principle, the number of possible process variations is determined by the degree of freedom the system has, i.e., the number of possible arrangements of different business contexts. A business process that is influenced by 6 business context elements $\{b_1, \ldots, b_6\}$, e.g., country, region, etc, that respectively have the following number of subtypes $\{8, 2, 5, 5, 3, 7\}$, will end up having more than 8000 variants. The immense management effort that is required to ensure the consistency of the process models in such a context is a very difficult and complex undertaking.

Figure 2(a) illustrates two variants of an eBay process model in the context of customer support. As the labels in the figure state that one of the models is called a `Parent process` and the other is called a `Child process`. A child process can reuse either parts of or the entire parent process. The terminology of child and parent is related to the inheritance concept, as the child (sub) process reuses behavior from the parent (super) process, similarly to how subclasses reuse (inherit) functionality from the super-classes. Any arbitrary process can be used as a parent process. Currently, a child process is de-

[3] `http://www.ebayinc.com/who`; June 6th 2010

rived by making a copy of the parent process and editing that copy, e.g., adding new activities or arbitrary control flow elements. Hence, there is no connectivity between the parent and the child processes. That is, a parent process could be edited by another modeler who might add new functionality without it being reflected on the child process, thus causing *inconsistency* between the child and parent processes. Manual maintenance of process models consistency is quite exhaustive and inefficient.

The notion of partial process model has been designed to address the above mentioned challenges [2,7]. It describes a desired process model through a combination of process model concrete elements and process views (queries). Figure 2(b) shows a partial process model that models how the `Child process` of Figure 2(a) can be obtained and maintained from the `Parent process`, depicted in the same figure. In Figure 2(b), the parts with grey background represent new activities that are introduced on the child process. To keep the relationship with inherited behavior, queries are used. $Q1$ keeps the link with activity `"take call"` from the parent process model. Also, $Q2$ keeps the relationship with the behavior of the parent process between activity `"ask for authentication data"` on the one hand and a termination possibility and activity `"ask for issue"` on the other hand. Thus, if the parent process behavior is changed by any means of, e.g., adding extra activities between the two activities, this is updated on the child process by reevaluating the queries against the parent process.

Once a partial process model is defined, it can be stored in the repository as a separate artifact that can be invoked in future. Indeed, there are two ways to invoke partial models. The first invocation is to view it. In this case, all queries in the partial model are matched to the respective parent processes. Matching parts are merged with concrete parts and the modeler is given an up-to-date view on how the child process looks like. In the view mode, the modeler might make changes to the process. In this case, if the change concerns overriding the behavior from the parent process, the modeler is warned and switched to the editing mode. The other invocation is to edit the partial process model. In that case, the modeler is allowed to arbitrarily edit query components or concrete components of the child process.

3 System Architecture

The architecture of the *BPModelMasher* framework is illustrated in Figure 3 and consists of the following main components.

- **Process Modeling, Querying, and Composition Environment** provides the process designer with a user-friendly *modeling interface* [5]. Users express their intention by means of a partial process model. The *query interface* extracts the set of process model queries from the partial process model, and passes them on to the query processor.
- **Process Model Repository.** Our framework can be connected to several, disparate, repositories to query process models that are stored remotely [10].
- **Uniform Language Interface** translates process models of specific languages to a common representation that is suitable to process model querying [10]. This allows to query a larger set of process models and can further be used to unify the query interface and processor toward different process definition languages [9].
- **Query Processor & Process Model Indexes.** The *query processor* evaluates the queries received from the query interface [10]. It provides support for the relax-

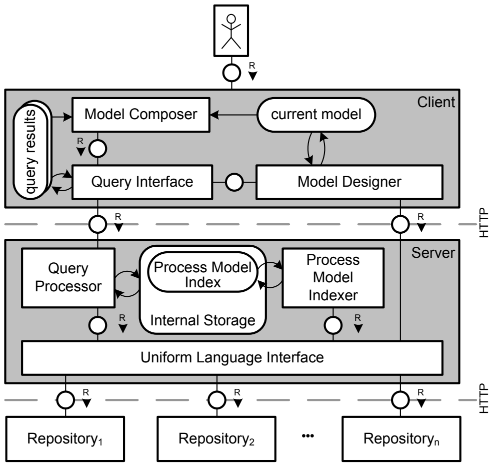


Fig. 3. Framework Architecture

ation and refinement of user queries. In case the queries do not return a sufficient results, the query processor is able to relax the query according to some similarity notions [3,6]. Similarly, if a query returns too many results, the user needs to be provided with the possibility of refining and improving their request. In order to further improve the searching, process models could be *indexed* upfront to expedite query evaluations.

The *model designer* component of our framework is the Oryx editor [5,10], an online extensible process modeling platform for research. The *query interface* and *query processor* for BPMN-Q [1] components have been implemented as a plugin to the Oryx editor and are able to run process model queries against the Oryx online process model repository. The *model composer* component is implemented as another plugin to the Oryx editor that uses the BPMN-Q query processor to evaluate the results of each query in the partial process model and then returns the composed process view to the end-user.

4 Demonstration

The demo will show that the business process modeling task can be very interactive and efficient using the proposed framework. In particular, we will demonstrate that the framework can improve the quality and the maturity of the process modeling task by reusing different process model components which are previously developed by business experts. Moreover, we will show how process designers can build different process views over the underlying process models and how the framework can maintain the consistency among these process variants automatically. The framework will be demonstrated using three different datasets: 1) The process model repository of *eBay* that has a set of innovative and special characteristics. 2) The dataset collected from the online business process modeling repository, ORYX. . 3) The dataset of the SAP reference model [4]. These datasets covers many application domains. Sample partial process models and process views will be ready to run, but users can visually edit and

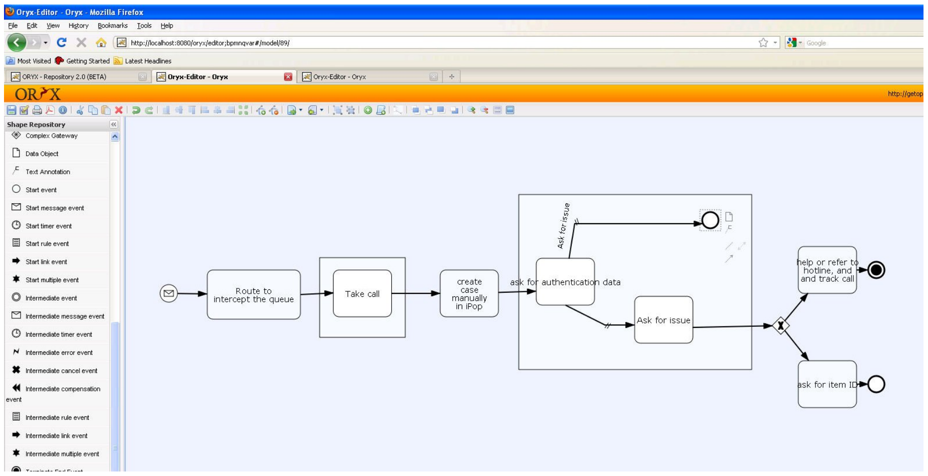

Oryx-Editor - Oryx - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/oryx/editor;bpmnqvar#/model/89/
ORYX - Repository 2.0 (BETA)
Oryx-Editor - Oryx
Oryx-Editor - Oryx
Shape Repository
Complex Gateway
Data Object
Text Annotation
Start event
Start message event
Start timer event
Start rule event
Start link event
Start multiple event
Intermediate event
Intermediate message event
Intermediate timer event
Intermediate error event
Intermediate cancel event
Intermediate compensation event
Intermediate rule event
Intermediate link event
Intermediate multiple event
Route to intercept the queue
Take call
create case manually in iPop
ask for authentication data
Ask for issue
Ask for issue
help or refer to hotline, and and track call
ask for item ID

Fig. 4. Screenshot of BPModelMasher design editor.

design their own ad-hoc views (see Figure 4 for a snapshot). The end users will also be able to view and validate the resulting process models for their ad-hoc definitions[2].

Besides the framework demonstration, we will discuss about the design choices that we have made on defining the granularity of process model components, similarity matching scores of process model components and the ranking process of the composed process models. In addition, the results of a user-study evaluation on the precision of the ranking of the composed process models over comprehensive datasets will be exhibited.

References

1. A. Awad. BPMN-Q: A Language to Query Business Processes. In *EMISA*, 2007.
2. A. Awad, M. Kunze, S. Sakr, and M. Weske. Design By Selection: A Reuse-based Approach for Business Process Modeling. In *ER*, 2011.
3. A. Awad, A. Polyvyanyy, and M. Weske. Semantic Querying of Business Process Models. In *EDOC*, pages 85–94, 2008.
4. T. Curran and G.Keller. *SAP R/3 Business Blueprint - Business Engineering mit den R/3-Referenzprozessen*. Addison-Wesley, 1999.
5. G. Decker, H. Overdick, and M. Weske. Oryx - sharing conceptual models on the web. In *ER*, pages 536–537, 2008.
6. R. Dijkman, M. Dumas, and L. García-Bañuelos. Graph Matching Algorithms for Business Process Model Similarity Search. In *BPM*, pages 48–63, 2009.
7. E. Pascalau, A. Awad, S. Sakr, and M. Weske. On Maintaining Consistency of Process Model Variants. In *BPM Workshops*, 2010.
8. E. Pascalau, A. Awad, S. Sakr, and M. Weske. Partial Process Models to Manage Business Process Variants. In *IJBPIM*, 2011.
9. M. La Rosa, H. Reijers, W. Aalst, R. Dijkman, J. Mendling, M. Dumas, and L. Garcia-Banuelos. Apromore: An advanced process model repository. *Expert Syst. Appl.*, 38(6):7029–7040, 2011.
10. S. Sakr and A. Awad. A Framework for Querying Graph-Based Business Process Models. In *WWW*, pages 1297–1300, 2010.

[2] Online demo and screen casts: http://bpmnq.sourceforge.net/BPModelMasher.html