Enacting Complex Data Dependencies from Activity-Centric Business Process Models

Andreas Meyer[1], Luise Pufahl[1], Dirk Fahland[2], and Mathias Weske[1]

[1] Hasso Plattner Institute at the University of Potsdam
{Andreas.Meyer,Luise.Pufahl,Mathias.Weske}@hpi.uni-potsdam.de
[2] Eindhoven University of Technology
d.fahland@tue.nl

Abstract. Execution of process models requires a process engine to handle control flow and data dependencies. While control flow is well supported in available activity-oriented process engines, data dependencies have to be specified manually in an error-prone and time-consuming work. In this paper, we present an extension to the process engine Activiti allowing to automatically extract complex data dependencies from process models and to enact the respecting models. We also briefly explain required extensions to BPMN to allow a model-driven approach for data dependency specification easing the process of data modeling.

Keywords: Process Modeling, Data Modeling, Process Enactment, BPMN, SQL

1 Introduction

Today, organizations use process-oriented systems to automate the enactment of their business processes. Therefore, processes are often captured in process models focusing on the activities being performed. These models are executed by process engines as, for instance, YAWL [1], Activiti [2], jBPM [6], Bonita [3], AristaFlow [8], and Adept2 [13]. Generally, a process engine has access to a process model repository as shown in Fig. 1. As soon as a start event of a particular process occurs, the engine creates a new instance of this process and enacts the control flow as specified by the process model. Thereby, the process engine is able to allocate specified user tasks to process participants via a graphical user interface or to invoke an application for execution of service tasks.

For the enactment of tasks, data plays an important role, because data specifies pre- and postconditions of tasks. A *precondition* requires the availability of certain data in a specified state while the *postcondition* demands certain manipulation of data. In modern activity-oriented process engines as mentioned above, these and further complex data dependencies (e.g., creating and updating multiplicity relations between data objects) have to be implemented manually through a process engineer by specifying the respective data access statements (see shaded elements in Fig. 1 left); this is an error-prone and time-consuming work.

In this paper, we explain an approach to *model* data dependencies in the process model itself and *automatically derive* data access statements from the process model as shown in Fig. 1 right. Process data utilized during activity execution is out of scope in this paper. We demonstrate the feasibility of this approach using the industry standard

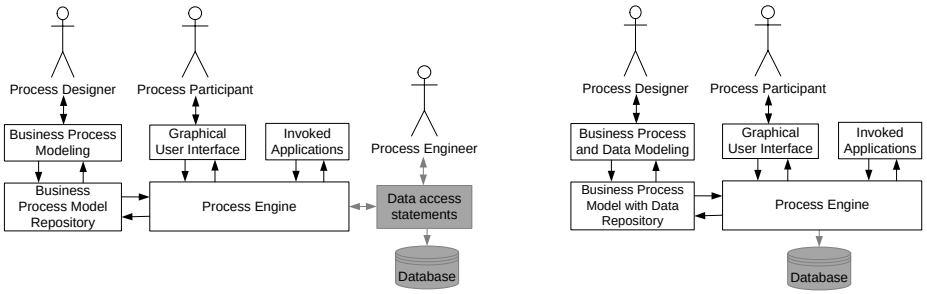

Fig. 1. Process engine architecture: classical (left) and proposed (right).

for process modeling, the Business Process Model and Notation (BPMN) [12], and the Activiti process engine [2].

Next, Section 2 shows how to model complex data dependencies in BPMN; Section 3 shows how three simple conservative extensions of the industrial process engine Activiti suffice to enact complex data dependencies from a BPMN model. We conclude in Section 4.

2 Modeling Complex Data Dependencies in BPMN

BPMN provides the concept of data objects that are associated to tasks [12]. Roughly, a task is only enabled when its associated *input* data objects are in a particular state. Associated *output* data objects have to be in a specified state when the task completes. However, for enactment more information is required.

Figure 2 shows a standard BPMN model of a simplified build-to-order process of a computer manufacturer (ignoring annotations set in italics). In this process, an *Order* that was *received* from a customer is first *Checked* and either *rejected* or *confirmed*. If it is confirmed, task *Create component list* creates several *Components* to be ordered; based on these components the order is processed in a subprocess and, when completed, the *Order* is *sent* to the customer.

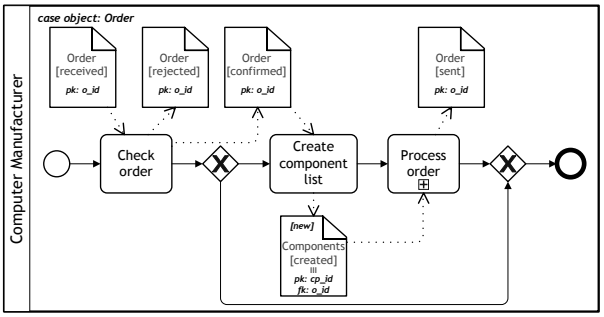

Fig. 2. Build-to-order Process of a Computer Manufacturer

In BPMN, each data object has a name and a set of attributes of which one describes the state of a data object. Data flow edges express pre- and postconditions to the different tasks, e.g., *Check order* is only enabled if object *Order* exists in state *received* in the current process instance. However, when handling multiple orders in different instances in parallel, the process model does not express which order is correlated to which process instance. Likewise, BPMN cannot describe *create*, *read*, *update*, and *delete* operations

on one or more objects of the same kind, possibly in 1:n or m:n relationship to other data objects. For instance, one cannot express that task *Create component list* of Fig. 2 creates several new Component objects and associates them to the Order handled in the process. Such data dependencies would have to be implemented manually.

In [9], we have shown that a few simple additional annotations to BPMN data objects suffice to describe such complex data dependencies with *operational semantics* directly in BPMN. First, borrowing an idea from business artifacts [5], we propose that for each process instance (and each instance of a subprocess) exists exactly one data object instance driving and orchestrating the execution of the process instance. All other data objects used in the instance depend on that *case object*. The case object of Fig. 2 is an *Order* as shown by the annotation. Dependencies between data objects are expressed via primary and foreign key attributes in analogy to databases [14]. Each data object has a primary key attribute that uniquely distinguishes different instances of this object, e.g., *Order* has the primary key attribute *o_id* and *Component* has *cp_id*. Foreign key attributes link object instances, e.g., attribute *o_id* links *Components* to *Orders*. The primary key of the case object implicitly links to the instance identifier of the (sub-)process. *Read* and *update* of data objects are already provided through BPMN's data flow edges. We express *create* or *delete* through respective annotations in the BPMN data object, e.g., *Create components list* creates several new *Components* (annotation *[new]* and multi instance characteristic |||) and relates them to the current *Order*.

Most importantly, these annotations have operational semantics. In [9], it is shown how to derive SQL queries from annotated BPMN data objects that realize the specified data operations. For example, for object *Order* in state *rejected* written by activity *Check order* in Fig. 2, the corresponding SQL query is dervied: `UPDATE Order SET state = 'rejected' WHERE o_id = $ID` with $ID representing the identifier of the (sub-)process the activity belongs to. See [9] for full details.

3 Tool Architecture and Implementation

We implemented the approach of Section 2 to show its feasibility. In the spirit of adding only few data annotations to BPMN, we made an existing BPMN process engine data-aware by only few additions to its control structures. As basis, we chose Activiti [2], a Java-based, lightweight, and open source process engine specifically tailored for a subset of BPMN. Activiti enacts process models given in the BPMN XML format. Activiti supports standard BPMN control flow constructs. Data dependencies are not enacted from the process model, but are specified separately. We adapted the Camunda [4] modeler to allow the creation of BPMN models with our proposed concepts; we extended the Activiti engine to enact process models with these concepts.

First, we extended the XML specification by utilizing *extension elements*, which the BPMN specification explicitly supports to add new attributes and properties to existing constructs. We added the *case object* as additional property to the (sub-)process construct. The data object was extended with additional properties for *primary key* (exactly one), *foreign keys* (arbitrary number), and the *data access type* as attribute. The BPMN parser of Activiti was extended to read BPMN data objects with the new attributes and properties, and data associations.

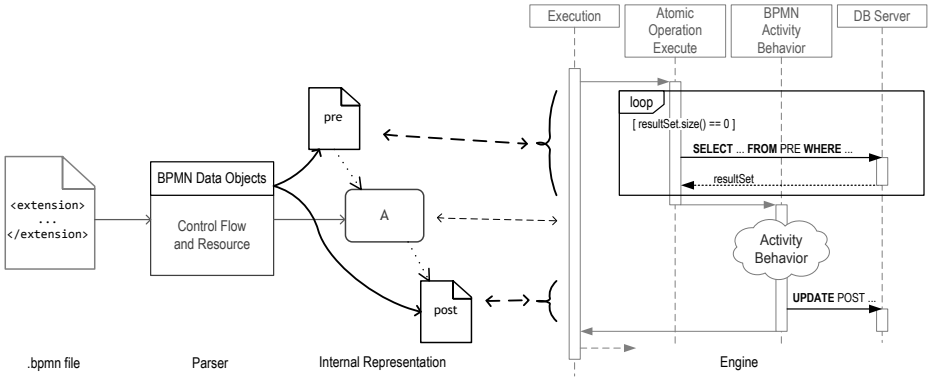

Fig. 3. Data dependencies are a conservative extension to data format, parser, internal representation, and execution engine.

The actual execution engine was extended at two points: before invoking the execution of an activity to check the preconditions of an activity and before completing an activity to realize the postconditions, both with respect to data objects. At either point, the engine checks for patterns of data input and output objects and categorizes them. For instance, in Fig. 2, *Order* is input and output to *Check order* in different states. The engine classifies this as a "*conditional update* of case object *Order*". The data operations at task *Create component list* would be classified as "*conditional creation* of multiple data objects that depend on the case object (1:n relationship)". Classification proceeds from most specific to most general patterns.

When invoking an activity, for each matching precondition pattern a corresponding SQL select query is generated to read whether the required data objects are available. The query assumes that for each data object of the process model exists a table holding all instances of this object and their attributes. If there is an object instance in the right state, the SQL query returns the corresponding entry and is empty otherwise. The engine repeatedly queries the database until an entry is returned (i.e., the task is enabled), as shown in Fig. 3. Then the activity is executed. Upon completion, an SQL insert, update, or delete query is generated for each matching postcondition pattern, and executed on the database.

Altogether, we had to extend Activiti at merely 4 points to realize our concept, as illustrated in Fig. 3: (1) at the XML, (2) at the parser and internal representation, (3) when checking for enabling of activities, and (4) when completing an activity. The extensions required just over 1000 lines of code with around 600 lines being concerned with classifying data access patterns, generating, and executing SQL queries. The extended engine, a graphical modeling tool, example process models, a screencast, and a complete setup in a virtual machine are available together with the source code at `http://bpt.hpi.uni-potsdam.de/Public/BPMNData`.

4 Conclusion

In this paper, we presented an approach how to automatically enact complex data dependencies from activity-centric process models. They key concepts required are

data objects associated to tasks; a few annotations allow to express relations between data objects and the particular data operation. Our modeling tool helps to easily specify the required annotations in a graphical user interface. From these annotations, SQL queries can be automatically generated and executed from a process engine, covering all fundamental data access operations: create, read, update, and delete of single data objects and of related data objects in 1:n and m:n relationship [9]. We have shown on the process engine Activiti that minimal extensions to existing execution engines suffice to implement this concept.

Compared to other techniques and engines for enacting data dependencies from models, our approach is less intrusive. The object-centric modeling paradigm [5, 11] requires substantial changes to the infrastructure as completely new engines have to be used. Process engines for this paradigm exist, e.g., PhilharmonicFlows [7] and Corepro [10], but they are incompatible with activity-centric approaches as it is supported by BPMN. In this respect, our work fills a critical gap in allowing owners of activity-centric processes to adapt automated enactment of data dependencies without changing paradigms and infrastructure.

Acknowledgements. We thank Kimon Batoulis, Sebastian Kruse, Thorben Lindhauer, and Thomas Stoff for extending the Camunda modeler [4] in the course of their master project to support the modeling of processes with respect to the concepts described in [9].

References

1. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language. Information Systems 30(4), 245–275 (2005)
2. Activiti: Activiti BPM Platform. `https://www.activiti.org/`
3. Bonitasoft: Bonita Process Engine. `https://www.bonitasoft.com/`
4. Camunda: Camunda BPM platform. `https://www.camunda.org/`
5. Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull. 32(3), 3–9 (2009)
6. JBoss: jBPM Process Engine. `https://www.jboss.org/jbpm/`
7. Künzle, V., Reichert, M.: PHILharmonicFlows: Towards a Framework for Object-aware Process Management. Journal of Software Maintenance and Evolution: Research and Practice 23(4), 205–244 (2011)
8. Lanz, A., Reichert, M., Dadam, P.: Robust and flexible error handling in the aristaflow bpm suite. In: CAiSE Forum 2010. vol. 72, pp. 174–189. Springer (2011)
9. Meyer, A., Pufahl, L., Fahland, D., Weske, M.: Modeling and Enacting Complex Data Dependencies in Business Processes. In: Business Process Management. LNCS, vol. 8094, pp. 171–186. Springer (2013)
10. Müller, D., Reichert, M., Herbst, J.: Data-driven modeling and coordination of large process structures. In: OTM 2007. LNCS, vol. 4803, pp. 131–149. Springer (2007)
11. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
12. OMG: Business Process Model and Notation (BPMN), Version 2.0 (2011)
13. Reichert, M., Rinderle-Ma, S., Dadam, P.: Flexibility in process-aware information systems. ToPNoC 5460, 115–135 (2009)
14. Silberschatz, A., Korth, H.F., Sudarshan, S.: Database System Concepts, 4th Edition. McGraw-Hill Book Company (2001)