Production Case Management: A Prototypical Process Engine to Execute Flexible Business Processes

Stephan Haarmann, Nikolai J. Podlesny, Marcin Hewelt,
Andreas Meyer, and Mathias Weske

Hasso Plattner Institute at the University of Potsdam
{Marcin.Hewelt,Andreas.Meyer,Mathias.Weske}@hpi.de

Abstract. In Business Process Management (BPM) recently several approaches have been proposed to handle flexible process execution, counter the strict adherence to process models at run-time, and allow for adaptations unforeseen at design-time. One of those approaches is Production Case Management (PCM) that aims at combining the strengths of structured process modeling and dynamic adaptation during run-time. PCM is an approach to model and execute flexible, data-driven, and user-centric business processes. Complementing the conceptual framework, we introduce a modeler and an engine for PCM in this paper.

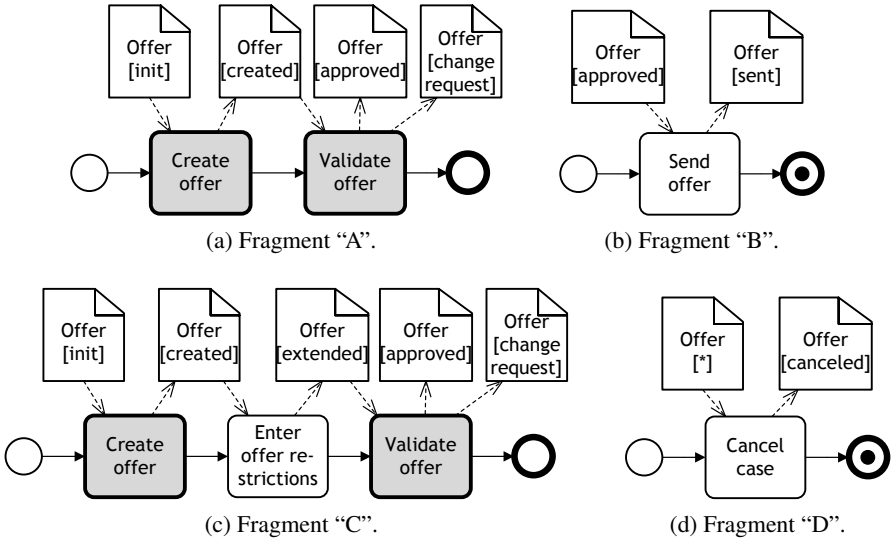
1 Introduction

Business Process Management (BPM) is an approach to systematically structure, support, and optimize the business operations of organizations. Usually, they are described by means of process models containing partially ordered control and data flow nodes and can be executed automatically by business process management systems [8]. In practice, process execution may deviate from process models due to unexpected happenings or because some employee found a more suitable way achieving the same goal. Targeting such situations, flexible process approaches have been introduced; one of them is Production Case Management (PCM) [4]. PCM builds upon the industry standard BPMN, the Business Process Model and Notation, and introduces design- and run-time flexibility. Thereby, PCM aims at combining the structure and guidance provided by traditional activity-centric BPM approaches, e.g., BPMN, with the flexibility of object-centric approaches like Adaptive Case Management (ACM) [7] and case handling [1]. While a conceptual framework for PCM has been initially described in [4], a concrete implementation of these concepts was left open.

In this paper, we present our prototypical implementation for PCM allowing modeling and execution of business processes represented as PCM scenarios. Therefore, we first summarize the PCM approach in Section 2. Afterwards, Section 3 presents the architecture of our tool, which is complemented by the information flow described in Section 4. Finally, Section 5 concludes the paper.

2 Production Case Management

Production Case Management (PCM) utilizes BPMN as process modeling language. But instead of using one large diagram for a single process model, PCM distributes the process logic over multiple smaller process fragments collectively defining the process model. Fragments may be added to a PCM scenario at any time, even during run-time, thus increasing the degree of freedom. At run-time, they are combined dynamically based on data dependencies. In fact, PCM distinguishes between control flow enablement and data enablement. An activity is control flow enabled, when the control flow reaches it; an activity is data flow enabled, when one specified input data set is completely available. An activity is *enabled*, if it is control-flow enabled *and* data-flow enabled. Additionally, the set of process fragments is accompanied by a data model together specifying a PCM scenario. Figure 1 shows four process fragments for an offer creation process adapted from a real-world process in [4], where fragments A and C show alternative ways of creating the offer and fragments B and D allow termination of the process model by either sending out or canceling the offer. In PCM, the termination event visualizes the termination condition, i.e. when a business process is completed; here: an offer object in state *canceled* or *sent*. Activities with a bold border are considered *link activities* meaning they are executed together in specific cases.



(a) Fragment "A". (b) Fragment "B".

(c) Fragment "C". (d) Fragment "D".

Fig. 1. Process fragments for "offer creation" example adapted from [4].

When a PCM scenario is instantiated, the start event of all process fragments occurs and the first control flow nodes, here four activities, get control flow enabled. Additionally, the data objects are initialized – usually in state `init`. However, since data objects may be used across process model boundaries, some objects may also be in some other state. In this paper, we assume an initialization in state *init*. Based on these data states, data availability is computed. In the given example, activities *Create offer* and *Cancel offer* are data flow enabled (the star * in the data node means that any state matches the

condition). Consequently, these activities may be executed since they are enabled from both control and data flow perspective.

Fragments are executed concurrently and repeatedly, i.e., once a fragment terminates a new instance of this fragment is created except if the termination condition is fullfilled. Then, all fragment instances terminate. Details on the operational semantics are provided in [4] and thus omitted here.

3 Architecture and Implementation

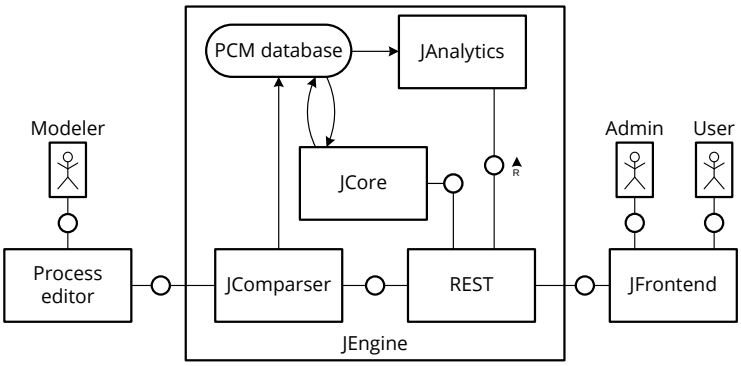


Fig. 2. JEngine Architecture as FMC block diagram [3].

Figure 2 shows the implemented architecture of our process engine in FMC notation[1] [3]. The architecture consists of two major modules communicating through a REST interface [2, 5]. The *JEngine* provides the functionality to execute PCM scenarios and allows to analyze the execution behavior through the extendable *JAnalytics* framework, based on which execution recommendations might be provided to the process participants. The *JFrontend*, the second main component, implements the REST calls and visualizes the enactment of the scenario in a dashboard. Targeting the modeling side of PCM, we adapted the open source editor[2] developed by inubit AG (now: Bosch Software Innovations GmbH) providing domain-model-compatible PCM scenarios. It contains a rich client and a server-sided part, which stores the created models in a repository that is accessible via a REST API. PCM scenarios from this repository can be imported into the JEngine via the *JComparser*. The editor can be easily replaced, as long as the XML serialization of PCM scenarios adheres to our domain model format.[3] Next, we briefly introduce the two main components.

JEngine. The *JEngine* consists of multiple sub-components. The *JComparser* fetches the process models, i.e., PCM scenarios, from the model repository of the process editor, deserializes the models following our domain model, and caches the information in

[1] Fundamental Modeling Concepts, `www.fmc-modeling.org/`

[2] `http://frapu.de/code/processeditor/`

[3] Details are given in the tool documentation at `https://bpt.hpi.uni-potsdam.de/Public/JEngineDoc`

Plain Old Java Objects (POJOs). Afterwards, they are persisted in the engine-internal *PCM database*, which is the only place where the models are referenced for the actual execution. Adaptations to the process models are also stored in the database that supports revision control to allow execution on old data sets if required by some process model. Generally, we migrate running process instances, the actual executions, to the latest data set, if only new process fragments were added. Otherwise, we preserve the old version with which the process instance was started.

This execution is handled by the *JCore* which encapsulates the corresponding logic based on the operational semantics given in the implementation framework in [4]. For the actual execution, scenarios, fragments, activities, and data objects are resources as defined in the REST specification. They are accessible through a unique URI. With regards to the CRUD schema [6], all resources can be used for interactions realized through the HTTP methods POST, GET, PUT and DELETE referring to create, read, update, and delete actions. Following the operational semantics, the *JCore* provides all currently enabled activities to the process participant who then chooses one for execution. Upon termination (or cancellation) of an activity, the *JCore* recomputes the enabled activities. If the next enabled activity is a web service or an email task – the two types of service tasks we currently support – it will be performed automatically by the *JCore*.

The *JAnalytics* sub-component stores each state change of an activity and data object including their attribute changes. Based thereon, we provide process monitoring capabilities and build-up a large event log for later statistical computations. This subcomponent provides a dynamic model-controller pattern for simply adding new algorithms and dynamic communication through REST for accessing the extended algorithms.

Frontend. The *JFrontend* implements the web interface for our PCM engine, thus allowing to monitor the process execution and to selectt the next activity to be executed. Our frontend is based on the widely used JavaScript library *AngularJS*[4]. To fetch the information to be displayed to the process participant and for PCM scenario deployment, execution, and configuration, the *JFrontend* utilizes the previously stated REST interfaces. The user interface provides two views: one for configuration and another one for execution. In the former, for instance, the service tasks are configured by specifying the web service to be accessed or the email specifics to be used for sending emails. The latter consists of fine-grained information-boxes that allow dynamic representation of the process progress. A detailed view on the user interface is provided in our screencast at `https://bpt.hpi.uni-potsdam.de/Public/JEngineDoc` while the upcoming section provides a walk-through of our tool.

4 Information Flow

Successful execution of process scenarios comprises five steps assuming that modeling is already completed and the scenarios are available in the process model repository.

[4] `https://angularjs.org/`

(1) The PCM scenarios are retrieved from the model repository and deployed through a REST call from the *JFrontend* to the *JComparser* and further to the process editor (chain of responsibility). Thereby, the process participant (user) selects the PCM scenarios to be fetched. Additionally, all corresponding process fragments are retrieved. Upon retrieval, the *JComparser* parses all fragments into the domain-model-format and enables persistence in the execution database. (2) Once imported, the *JFrontend* can execute the scenario through the *JCore*. Requests from the user (manual task) or from the *JCore* (service task) allow the start and termination of process instances and activities as well as manipulation of data objects. (3) All changes are saved in the history (*JAnalytics*) and can be accessed through corresponding REST interface. (4) While analysis can be performed based on historic and current data through our analysis framework, there is no generic user interface for the *JAnalytics* framework such that new algorithms must be integrated manually. (5) Finally, all information is returned to the *JFrontend* and visualized to the process participant.

5 Conclusion

In this paper, we presented our prototypical implementation of a process engine to execute process models following the concept of Production Case Management (PCM). PCM is a novel approach to allow design-time and run-time flexibility for BPMN while benefiting from BPMN's acceptance and preserving its idea of structural and guidance-based model enactment. The publicly available source code, the documentation, and a screencast of our process engine are available at `https://bpt.hpi.uni-potsdam.de/Public/JEngineDoc`.

Acknowledgements. We thank Jaspar Mang, Juliane Imme, Jan Selke, and Sven Ihde for their continuous support towards implementation of this prototypical process engine for PCM. Additinally, we thank Frank Puhlmann for many fruitful discussions and Bosch Software Innovations GmbH for funding the connected research project.

References

1. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case Handling: A New Paradigm for Business Process Support. Data & Knowledge Engineering 53(2), 129–162 (2005)
2. Fielding, R.T., Taylor, R.N.: Principled Design of the Modern Web Architecture. ACM Transactions on Internet Technology (TOIT) 2(2), 115–150 (2002)
3. Knöpfel, A., Gröne, B., Tabeling, P.: Fundamental Modeling Concepts. Wiley (2005)
4. Meyer, A., Herzberg, N., Puhlmann, F., Weske, M.: Implementation Framework for Production Case Management: Modeling and Execution. In: EDOC. pp. 190–199 (2014)
5. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly Media, Inc. (2007)
6. Silberschatz, A., Korth, H.F., Sudarshan, S.: Database System Concepts. McGraw-Hill Book Company (2010)
7. Swenson, K., Palmer, N., Pucher, M.: Case Management: Contrasting Production vs. Adaptive. In: How Knowledge Workers Get Things Done. pp. 109–118 (2012)
8. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Second Edition. Springer (2012)