Leonardo Azevedo, Cristina Cabanillas (Eds.)

BPM 2016 Demonstration Track

Demonstration Track of the
14th International Conference on
Business Process Management (BPM 2016)
Rio de Janeiro, September 21, 2016

Proceedings

Volume Editors

Leonardo Guerreiro Azevedo
Federal University of the State of Rio de Janeiro (UNIRIO)
Av. Pasteur, 458 - Urca
22290-240 Rio de Janeiro, Brazil
azevedo@uniriotec.br

IBM Research Brazil
Av. Pasteur, 146 & 138 - Botafogo
22290-240 Rio de Janeiro, Brazil
lga@br.ibm.com

Cristina Cabanillas
Vienna University of Economics and Business (WU)
Welthandelsplatz 1
1020 Vienna, Austria
cristina.cabanillas@wu.ac.at

Preface

The International Conference on Business Process Management (BPM) brings together researchers and practitioners and represents one of the most prestigious scientific events on BPM worldwide. The 14th edition of the conference was organized in Rio de Janeiro, Brazil, on September 18-22, 2016. The venue took place in the Othon Palace Rio Hotel, situated in front of one of the most famous and beautiful beaches in the world (Copacabana Beach), which created a perfect atmosphere.

As a novelty with respect to previous BPM Demo track editions, this year all the demonstration proposals submitted had to include a screencast showing the functionality of the tool. The BPM 2016 Demo track received 20 submissions, which were carefully reviewed by an international Program Committee composed of 36 members. Out of them 14 proposals were selected for presentation at the conference. The authors of the accepted demos were given the opportunity to also bring a poster to the demo session in order to attract audience and increase the visibility of their tool. Furthermore, after the conference, they could upload their tool to the BPM Tool Database[1].

The demo session took place on September 21. Two Demo Teaser sessions were organized simultaneously with conference coffee breaks to advertise the demos that would be presented later. The 14 accepted demos were distributed between the two time slots and the authors of each demo had 2 minutes to briefly describe their tool. The actual Demo session took place in the afternoon and was organized in the same open space where the conference coffee breaks took place, which we believe had a positive influence in the amount of audience attending the session. We are glad that some attendees stayed there until the end of the session, which showed a good acceptance of the track. The attendees could contribute to the selection of the best demo by providing anonymous votes.

The BPM 2016 Demo Award was given at the conference banquet together with the other BPM 2016 awards. The tool "PLG2: Multiperspective Process Randomization with Online and Offline Simulations" by Andrea Burattin was selected as the best BPM 2016 demo. We congratulate the author for this achievement!

We would like to thank all the authors for their submissions, the members of the Program Committee for their time, their hard work and the timely submission of their reviews, and the organizers of the BPM 2016 conference for their great support, which made this demo track possible.

Rio de Janeiro
September 2016

Leonardo Azevedo
Cristina Cabanillas

[1] Available online at `http://bpm-conference.org/bpt-resource-management/`

Organization

Demo Chairs

Leonardo G. Azevedo	Federal University of Rio de Janeiro State, Brazil IBM Research Brazil, Brazil
Cristina Cabanillas	Vienna University of Economics and Business, Austria

Program Committee Members

Michael Adams	Queensland University of Technology, Australia
Fernanda Araujo Baiao	UNIRIO, Brazil
Claudia Cappelli	UNIRIO. Brazil
Jorge Cardoso	University of Coimbra, Portugal
Jan Claes	Ghent University, Belgium
Raffaele Conforti	Queensland University of Technology, Australia
Laurent D'Orazio	UBP, France
Gero Decker	Signavio, Germany
Remco Dijkman	Eindhoven University of Technology, Netherlands
Marcelo Fantinato	University of So Paulo (USP), Brazil
Marie-Christine Fauvet	Joseph Fourier University of Grenoble, France
Juliana Jansen Ferreira	PUC-RIO, Brazil
Christian Gierds	Carmeq GmbH, Germany
Oliver Kopp	IAAS, University of Stuttgart, Germany
Geetika Lakshmanan	IBM T J Watson Research Center, USA
Henrik Leopold	VU University Amsterdam, Netherlands
Heiko Ludwig	IBM Research, USA
Andrea Magdaleno	Fluminense Federal University (UFF), Brazil
Cesare Pautasso	University of Lugano, Switzerland
Artem Polyvyanyy	Queensland University of Technology, Australia
Hajo A. Reijers	Eindhoven University of Technology, Netherlands
Stefanie Rinderle-Ma	University of Vienna, Austria
António Rito Silva	IST/INESC-ID, Portugal
Carlos Rodriguez	University of Trento, Italy
Anne Rozinat	Fluxicon, Netherlands
Nick Russell	Queensland University of Technology, Australia
Sherif Sakr	The University of New South Wales, Australia
Vishal Saxena	Roubroo, USA
Stefan Schönig	University of Bayreuth, Germany
Jianwen Su	University of California at Santa Barbara, USA
Roman Vaculín	IBM Research, USA
Boudewijn Van Dongen	Eindhoven University of Technology, Netherlands
Ingo Weber	NICTA, Australia
Mathias Weske	HPI, University of Potsdam, Germany
Karsten Wolf	Universität Rostock, Germany
Moe Wynn	Queensland University of Technology, Australia

Table of Contents

Demo Papers

PLG2: Multiperspective Process Randomization with Online and Offline Simulations

Andrea Burattin
University of Innsbruck, Austria
andrea.burattin@uibk.ac.at

Abstract. The evaluation of process mining algorithms requires, as any other data mining task, the availability of large amount of (real-world) data. Despite the increasing availability of such datasets, they are affected by many limitations: *in primis*, the absence of a "gold standard" (i.e., the reference model). This work extends an approach already available in the literature for the generation of random processes. Novelties have been introduced throughout the work which, in particular, involve the complete support for multiperspective models and logs (i.e., the control-flow perspective is enriched with time and data information) and for online settings (i.e., generation of multiperspective event streams and concept drifts). The proposed new framework is able to cover the spectrum of possible scenarios that can be observed in the real-world.

Keywords: Process mining; log generation; event stream; concept drift.

1 Introduction

Process mining [1] gained a lot of attention and is now considered an important field of research, bridging data mining and business process modeling/analysis. Particularly, the aim of process mining is to extract useful information from business process executions. In data mining, the term *gold standard* (also referred to as *ground truth*) typically indicates the "correct" answer to a mining task (i.e., the reference model). For example, in data clustering, the gold standard may represent the right (i.e., the target) assignment of elements to their corresponding clusters. Many times, referring to a gold standard is fundamental in order to properly evaluate the quality of mining algorithms. Several concepts, like *precision* or *recall*, are actually grounded on this idea. However in the context of business processes, companies are usually reluctant to publicly share their data for analysis purposes. Moreover, detailed information on their running processes (i.e., the reference models) are considered as company assets and, therefore, are kept even more private. An annual event, called *BPI challenge*, releases real world event logs. Despite the importance of this data, the logs are not accompanied with their corresponding gold standards. Moreover, they do not provide examples of all possible real world situations: many times, researchers and practitioners would like to test their algorithms and systems against specific conditions and, to this purpose, those event logs may not be enough.

In this paper we propose the new version of a tool already available [5] (PLG). The new tool, Processes and Logs Generator 2 (PLG2), can be used to randomly generate multiperspective process models and to simulate them, with the purpose of synthesizing multiperspective event logs. Moreover, the approach is tailored to the simulation of online settings, since it is possible to generate local evolutions of processes and event streams. Another feature allows to dynamically change the model underlying the stream, in order to produce concept drifts. PLG just allowed the generation of random control-flows and their simulation as static logs. PLG2 is implemented in a standalone Java application which is also accompanied by a set of APIs, useful for the programmatic definition of custom experiments.

The implemented components are reported in Fig. 1. The central part is responsible for the representation of a process model. In order to create a process, the user can import a file; randomly generate a new process; or evolve an existing process into a different one. Processes can also be exported to file. Starting from a process model, PLG2 can simulate it in order to create an event log or an event stream. Both these last components use a noise generator in order to add more realism to the generated data.

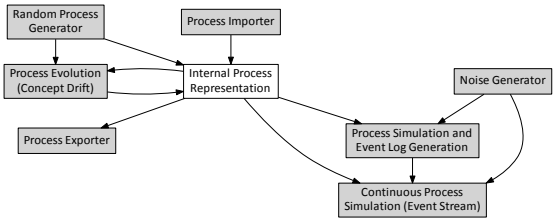


Fig. 1: Components of PLG2.

2 PLG2: Technical Details

Process Models. In PLG2, the internal structure of a process model is intuitively derived from the definition of a BPMN process model. A process is essentially an aggregation of *components*. Each component can be either a flow object, a sequence or a data object. Flow objects are: events (*start* or *end*); gateways (*exclusive* or *parallel*); and tasks. Sequences connect two flow objects. Data objects are associated with activities and can be *plain* or *dynamic*. A plain data object is a name-value pair. A dynamic data object has a value that can change every time it is required (i.e., it is dynamically generated by a script). Data objects can be *generated* or *required* by activities. This internal structure allows more flexibility. For example, it is now possible to load BPMN models generated with external tools, as long as the modeled components are available also in PLG2. Also, since we are restricting to non-ambiguous components, we can convert our processes into Petri nets. The generation of random processes is based on some workflow control-flow patters [2]: *(a)* WCP-1: direct succession of two activities; *(b)* WCP-2: parallel execution; *(c)* WCP-3: synchronization of parallel branches; *(d)* WCP-4: mutual execution; *(e)* WCP-5: convergence of branches; *(f)* WCP-21: ability to execute sub-processes repeatedly. By progressively combining these patterns we build a complete process model. The combination of these patterns is performed according to a predefined set of probabilistic rules.

Detailed description of the generation rules is reported in the technical report [3]. Random data objects are generated and randomly connected to activities as well.

Process Simulation. The procedure for the generation of logs starting from business processes, basically, consists of a simulation engine running the "*play-out game*": the algorithm keeps a set of "enabled activities" and randomly picks one for the simulation (different probabilities are not supported yet). After that, it checks for new enabled activities, puts them into the enabled activities set, and repeats itself. To determine activities times and durations, the system checks for any of these parameters. If these are not reported, the activity is assumed to be instantaneous and to execute just after the previous. However, the user can specify these parameters as Python functions: `time_after(caseId)` and `time_lasted(caseId)`. Both functions are called by the simulator with the `caseId` parameter valued with the actual case id: this allows the functions to be case-dependent. `time_after(caseId)` returns the number of seconds to wait before the following activity starts. `time_ lasted(caseId)` returns the number of seconds that the activity lasts. This approach is extremely flexible: it is possible to define, for example, different durations for the same activity depending on which flow the current trace has followed. Regarding data objects, *generated* data objects are stored as attribute of the current activity, *required* are written as attributes for the preceding activity. *Plain* data objects are treated as fixed values (i.e., the simulation generates always the same value); *dynamic* data objects are actually Python scripts whose values are determined by the execution of the script itself. These scripts implement a `generate(caseId)` function which returns an integer or a string value. Please note that, also in this case, the function is called with the `caseId` parameter valued with the actual instance case id, providing the user with an in-depth, and case dependent, control over the generated values. There is no particular limit on the number of plain and dynamic data objects that a task can have, both required and generated. It is also possible to introduce simulated noise. Noise can be at different "levels": *(i)* at the trace level (involving trace organization); *(ii)* at the event level (involving events on the control-flow); *(iii)* at the data object level (involving the data perspective). The actual noise generation is driven by the parameters set by the user. The noise details for the trace and the event level have already been discussed in the literature and reported in details in [7]. The remaining technical details are described in [3].

PLG2 is explicitly design for the simulation of online event streams [6]: the *play-out game* should continue infinitely and the underlying process model is allow to change. The stream definitions used are reported in [4, 6]. An event stream is just a sequence of events, each of them potentially belonging to different traces. From an implementation point of view, the idea is to create a socket, which is accepting connections from external clients. PLG2, then, "emits" (i.e., writes on the socket) the generated events. In order to generate a continuous stream, the user has to set two parameters: the maximum number of parallel instances running at the same time and the "time scale". The first parameter is

(a) Starting process model. (b) First evolution. (c) Second evolution.

Fig. 2: A process model randomly generated with two sequential evolutions.

used to populate the data structures required for the generation of the stream. Then, since the event emission is performed in "the actual time" (opposed to the "simulated time"), it might be necessary to scale the simulation time in order to have the desired events emission rate. To this end we need a time multiplier, which is expected to be defined in $(0, \infty]$. This time multiplier is used to transform the duration of a trace (and the time position of all the contained events), from the simulation time to the real time.

One typical aspect of online settings is the presence of concept drifts. The tool is able to dynamically switch the source generating the events but to change the stream source, a second model is required. To create another model, two options are available: one is to load or generate from scratch a model; the other is to "evolve" an existing one: this is an important feature of PLG2. To evolve an existing model, PLG2 replaces an activity with a subprocess generated using the random procedure already used for the process randomization. The new process could be very similar to the originating one or very different, and this basically depends on the probability configured by the user. For example, Fig. 2 reports two evolutions of the process model, which has been randomly generated. Please note that evolutions can involve the creation or the deletion of data objects as well.

Implementation The tool is implemented as a Java application. It is available as open source project and also binary files are provided.[2] The project APIs can also be easily used to randomly generate processes or logs. The current implementation is able to load BPMN files generated with Signavio or PLG2. Model can be exported as PNML or PLG2 file, or as graphical representation (as BPMN and Petri net) using the Graphviz file format. The simulation of log files generates a XES-compliant objects, which can be exported both as XES or MXML (for compatibility with ProM 5). From Fig. 3 it is possible to see the main structure of the GUI: there is a list of generated processes on the left. The selected process is shown on the main area. Right clicking on activities allows the user to set up activity-specific properties (such as times, or data objects). On the bottom part of the main application it is possible to see the PLG2 console. Here the application reports all log information, useful for debugging purposes. The application dialog in the foreground is used for the configuration of the Python script which will be used to determine the time properties. As shown, specific syntax highlighting and other typing hints (such as automatic

[2] See http://plg.processmining.it and https://github.com/delas/plg.

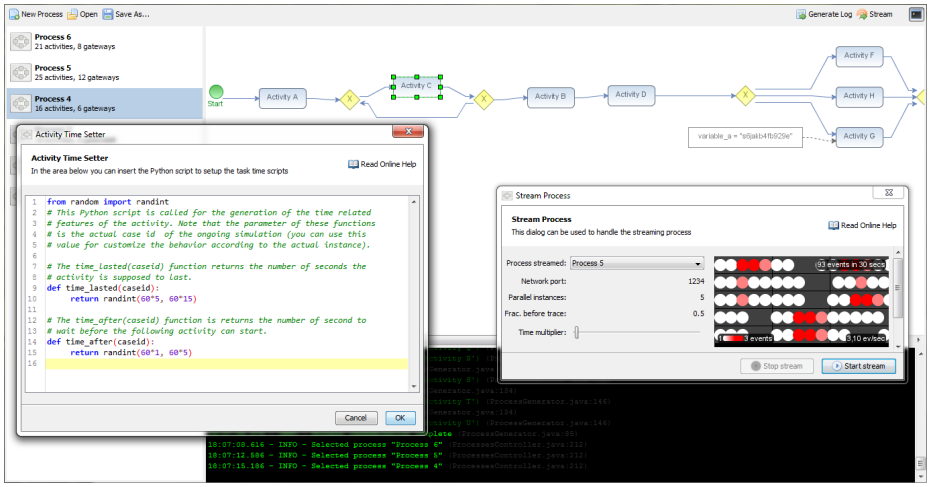


Fig. 3: Screenshot of PLG2 with several models in the workspace; the time setting dialog for "Activity C", and the log console. The stream dialog is also displayed.

indentation) helps the user in writing Python code. The stream dialog is also displayed in foreground: it is possible to dynamically change the streamed process (using the "Process streamed" combo box) and the time multiplier. The right hand side of such dialog (in the rectangle with black background) reports "a preview" of the stream: 30 seconds of the stream are reported and each filled circle represents, in this case, up to 3 events.

3 Conclusion

This paper describes PLG2, which is the evolution of an already available tool. While that tool was able to randomly generate process models and simulate them, PLG2 extends the support to multiperspective models (by adding detailed control of time perspective and introducing data objects) and has full support for the simulation of offline and online settings (generating drifting models and simulating event streams). A screencast showing PLG2 features is available at http://youtu.be/t-GMV4hU_vs.

References

1. van der Aalst, W.M.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer Berlin / Heidelberg (2011)
2. van der Aalst, W.M., ter Hofstede, A.H., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
3. Burattin, A.: PLG2: multiperspective processes randomization and simulation for online and offline settings. CoRR abs/1506.08415 (2015)
4. Burattin, A.: Process Mining Techniques in Business Environments. Springer (2015)

5. Burattin, A., Sperduti, A.: PLG: a Framework for the Generation of Business Process Models and their Execution Logs. In: Proceedings of BPI. Springer (2010)
6. Burattin, A., Sperduti, A., van der Aalst, W.M.: Control-flow Discovery from Event Streams. In: Proceedings of the IEEE CEC (2014)
7. Günther, C.W.: Process mining in Flexible Environments. TU Eindhoven (2009)

The DCR Graphs Process Portal

Søren Debois[1], Thomas Hildebrandt[1], Morten Marquard[2], and Tijs Slaats[3] *

[1] IT University of Copenhagen, Denmark
{debois, hilde}@itu.dk, http://www.itu.dk
[2] Exformatics A/S, Denmark
{mmq}@exformatics.com, http://www.exformatics.com
[3] Department of Computer Science, University of Copenhagen, Denmark
{slaats}@di.ku.dk, http://www.diku.dk/

Abstract. We demonstrate the dcrgraphs.net Process Portal: a cloud-based solution for collaborative continuous development and analysis of knowledge intensive processes. This tool is the result of a long-term collaboration between Exformatics A/S, a Danish provider of Adaptive Case Management (ACM) solutions, and researchers from IT University of Copenhagen (ITU) and the Department of Computer Science at University of Copenhagen (DIKU). The tool draws heavily on current research into declarative process modelling notations: it is built upon the declarative Dynamic Condition Response (DCR) Graphs notation. In our demonstration we will introduce the primary features of the tool: (1) a portal for creating, storing and sharing processes, (2) a declarative process designer, (3) a collaborative simulator, and (4) process analysis modules. The intended audience are researchers and practitioners interested in process modelling notations and techniques for knowledge workflow management and similarly flexible process environments.

Introduction In this demonstration we present dcrgraphs.net, a cloud-based commercial solution for modelling, simulating and analysing declarative process models, built and supported commercially by Exformatics A/S. The solution is free for academic use.

The core philosophy of the tool is that the declarative and imperative modelling paradigms are solving fundamentally distinct problems, and therefore tooling around them must be similarly distinct. Declarative modelling notations, such as Declare [6] and Dynamic Condition Response (DCR) Graphs [2, 4, 8] achieve flexibility by encoding a multitude of potential executions of a workflow into a set of rules. This means on the one hand that a single model embodies potentially orders of magnitudes more potential workflow executions—more flexibility!—than imperative notations such as BPMN [5], but on the other that the modeller has to account for the increased complexity that comes with these many different possible executions. Thus, declarative models

* Authors listed alphabetically. This work is supported in part by the Hybrid Business Process Management Technologies project (DFF 6111-00337, 2016-2019) and the Computational Artifacts project (VELUX 33295, 2014-2017)

buy flexibility at the cost of complexity. A declarative model is not as readily understood from "looking at the model" [7] as is a model in an imperative notation such as BPMN [5], but this is to be expected: imperative models express less complexity. They solve a different problem than declarative models.

The tool is grounded in the belief that flexibility is a necessity for Adaptive Case Management, and that this complexity is a necessary evil of that flexibility. The only way forward is to aid the modeller in creating and understanding his model. *Providing this support is the core aim of the* `dcrgraphs.net` *solution.*

The portal, its main features, and the underlying DCR Graph notation were reported on in [3]. The formal semantics of the notation and some of its primary extensions are described in [2, 4, 8].

In the present paper we describe the main features of the portal and how they help curb complexity:

1. The main portal, which acts as a social repository for declarative process models in which models can be shared and discussed with other users.
2. The process designer, which enables the user to model their processes.
3. The simulator, which supports both individual and collaborative simulation
4. Analysis tools, which allow the user to understand model behaviour.

At the live demonstration we will familiarise the audience with each of these features by creating, modelling, simulating and analysing an example model. Audience members will be invited to participate in collaborative aspects from their own devices.

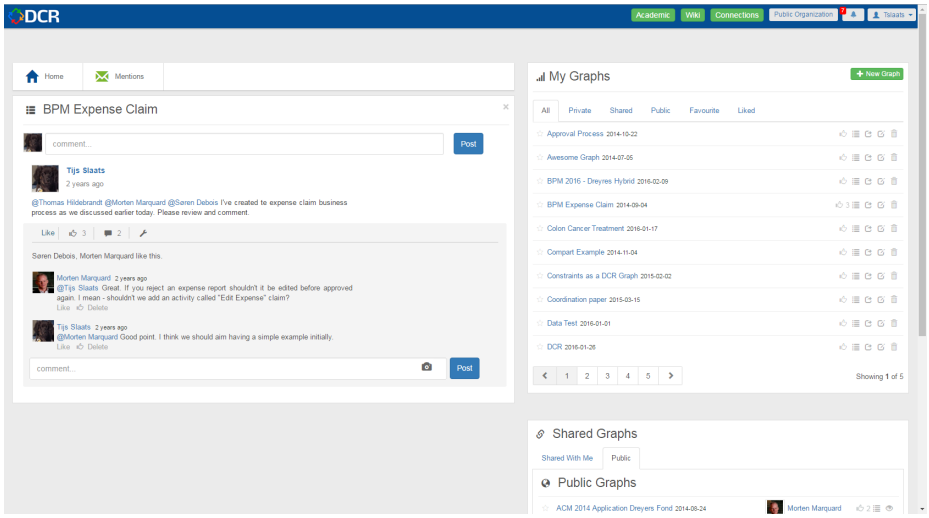


Fig. 1. The Process Portal

Process Portal Figure 1 shows the main portal. The right half of the screen is used to display lists of personal, shared and public process models. The left half of the screen

displays *activity streams*, which can be used to discuss with other users. Users can add colleagues and friends to their personal network, which allows them to share models and join collaborative simulations.

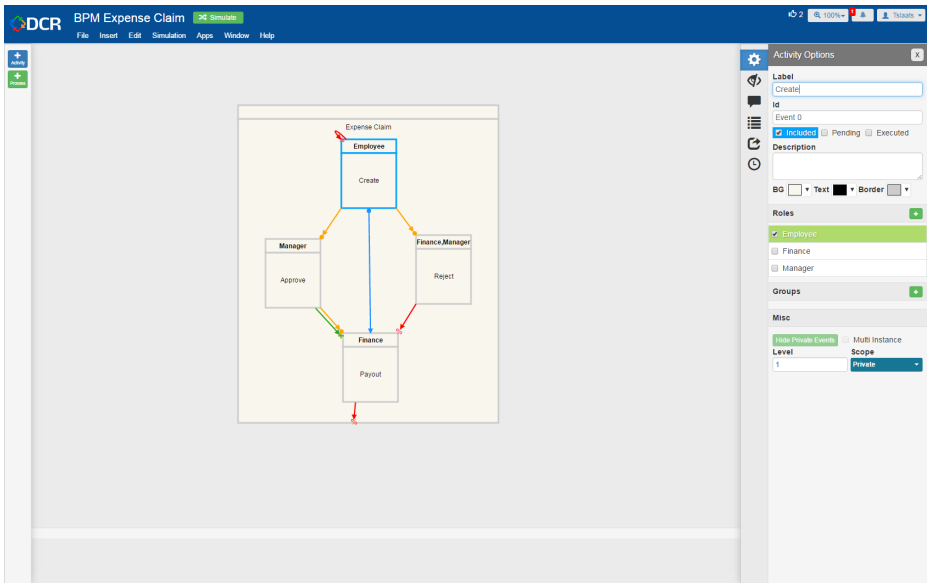


Fig. 2. The Process Designer

Process Designer Figure 2 shows the process designer. At the centre is the process model, consisting of boxes representing activities and relations between the boxes that represent the rules governing the process. Clicking on activities and rules brings up an options pane which allows one to customise that element. To facilitate the understandability of process models a number of visual filters are offered by the designer: activities can be assigned a *level of abstraction* at which they are shown, meaning that the users can start to view the process at the highest level of abstraction, showing only a few activities and then gradually introduce more details by lowering the level of abstraction. In addition activities can be assigned to any number of *user-defined groups*, which e.g. allows the user to view only those activities that are relevant to a particular organisational unit. The designer also supports fine-grained revision tracking, which allows any change to the model to be inspected and reverted. The designer can import and export DCR Graphs in a standardised XML format which is used by a number of external tools.

Collaborative Simulation Many processes involve a collaboration between multiple actors, often in different organisations. For the successful implementation of such processes it is imperative that the end-users understand their own role. To facilitate this the DCR Portal supports both individual and collaborative simulation, which can be started

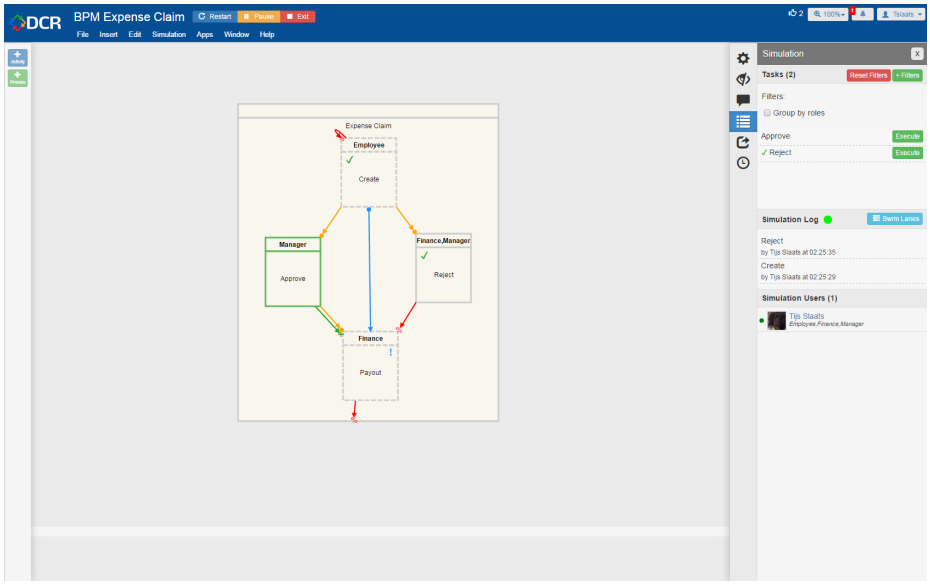


Fig. 3. The Simulation Window

from the designer. Here the user can invite connections to their simulation and assign roles to them. Figure 3 shows the simulation window; on the left one can see the current DCR Graph, with its marking updated according to the actions taken by the users. On the right the user is presented with a task list, the log of previously executed tasks and a list of participants.

Analysis Tools The DCR Portal offers a flexible plug-in framework which can be used by third parties to create additional functionality for the portal. As a proof-of-concept several analysis tools have been developed as plug-ins. The first is the *path analyser*, essentially a GPS for processes: the user can enter a start and goal activity and the analyser will determine the shortest possible execution path to get to the goal. Typical GPS-like functions, such as intermediate activities or activities to avoid are also supported. The other plug-in is the *dead-end analyser*, which can be used to detect dead- and live-lock in processes. Figure 4 shows an example of the path analyser plug-in. Here the user has asked for a path from *Create* to *Payout*, using the activity *Reject* at least once. The proposed path is drawn as a swimlane diagram in the bottom window.

Maturity, Availability, Documentation and Video Tutorials Altogether, the development of DCR Graph technologies and tools started in 2010 [9]. The development of the portal in its present form [3] started in the spring of 2014 and the first official release was in the autumn of the same year. It has since been continuously updated.

Since the spring of 2015 the portal has been used in classes at IT university of Copenhagen, most prominently a process modelling class for 75 bachelor and M.Sc. students. The portal has also been used for teaching at the Federal University of the

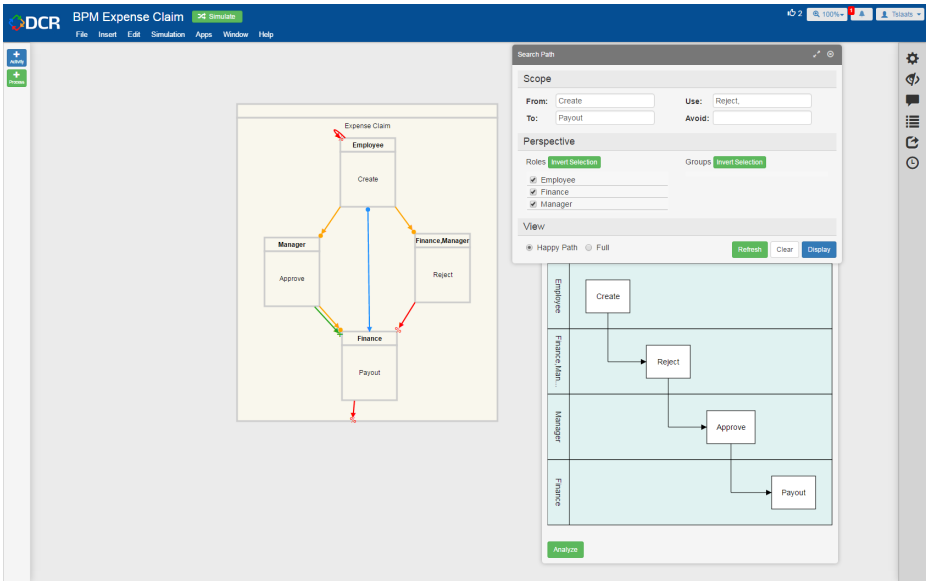


Fig. 4. Process Analysis

State of Rio de Janeiro. Exformatics uses the portal to model and maintain the processes of their customers, in particular a Danish foundation [1], whose Adaptive Case Management solution uses a declarative process engine for executing its processes.

The portal is accessible for free online at `http://www.dcrgraphs.net/`. Documentation is available through a wiki at `http://wiki.dcrgraphs.net/`, which contains video tutorials explaining how to use the different features of the portal.

References

1. S. Debois, T. Hildebrandt, M. Marquard, and T. Slaats. A case for declarative process modelling: Agile development of a grant application system. In *AdaptiveCM '14*.
2. S. Debois, T. Hildebrandt, and T. Slaats. Hierarchical declarative modelling with refinement and sub-processes. In *BPM '14*, volume 8659 of *LNCS*, pages 18–33.
3. M. Marquard, M. Shahzad, and T. Slaats. Web-based modelling and collaborative simulation of declarative processes. In *BPM '15*.
4. R. R. Mukkamala. *A Formal Model For Declarative Workflows - Dynamic Condition Response Graphs*. PhD thesis, IT University of Copenhagen, March 2012.
5. OMG BPMN Technical Committee. Business Process Model and Notation, version 2.0.
6. M. Pesic, H. Schonenberg, and W.M.P. van der Aalst. DECLARE: Full Support for Loosely-Structured Processes. In *EDOC '07*, pages 287–. IEEE.
7. P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, and H. A. Reijers. Imperative versus declarative process modeling languages: An empirical investigation. In *Business Process Management Workshops*, volume 99 of *LNBIP*, pages 383–394. Springer, 2012.
8. T. Slaats. *Flexible Process Notations for Cross-organizational Case Management Systems*. PhD thesis, IT University of Copenhagen, January 2015.
9. T. Slaats, R. R. Mukkamala, T. Hildebrandt, and M. Marquard. Exformatics declarative case management workflows as dcr graphs. In *BPM '13*, volume 8094 of *LNCS*, pages 339–354.

MuDePS: Multi-perspective Declarative Process Simulation

Lars Ackermann and Stefan Schönig

University of Bayreuth,
Universitätsstraße 30, 95447 Bayreuth, Germany
{lars.ackermann,stefan.schoenig}@uni-bayreuth.de

Abstract. Business process simulation supports the improvement and analysis of business process models. Especially log generation features gain more and more attractivity, for instance, because of their applications in the evaluation of process mining techniques. Additionally, cognitive science has shown that examples promote the comprehension of abstract models. This is important especially for declarative modeling languages, which are able to cope with highly flexible processes while mostly being less intelligible than imperative counterparts. Since most of the available process simulation tools are tailored to imperative languages they are not suitable in declarative contexts. The tool presented in this paper is able to simulate declarative, multi-perspective process models. It builds on DPIL, a declarative process modeling language. DPIL models can be created and compiled to a simulation model using the DPIL Modeler. MuDePS, a second component, is able to generate event logs that conform to the original DPIL model. Each generated log is able to describe an exhaustive, distinct set of valid process execution traces of a desired length.

Keywords: process simulation, log generation, declarative modeling

1 Background and Significance to BPM

Business processes differ regarding their strictness and level of governance. Thus, two paradigms for business process modeling emerged: Imperative models describe allowed process instances using step-by-step flow descriptions, whereas declarative, i.e. rule-based models consist of constraints that each instance has to satisfy [1]. Declarative process modeling languages (DPMLs) are based on the foundational principle that all execution paths are allowed as long as they are not explicitly forbidden. Rules are used to form a forbidden region for process executions. Hence, more flexible processes require less rules, i.e., the resulting model is comparably small and, thus, rule-based approaches are well-suited for the representation of flexible processes [1, 2].

```
use group Administration

process BusinessTrip {
    task Apply for Trip
    task Approve application
    task Book accommodation

    document Application

    ensure produces(Apply for Trip, Application)
    ensure consumes(Approve application, Application)
    ensure sequence(Approve application, Book accommodation)
    ensure role(Approve Application, Administration)
    ensure binding(Book accommodation, Apply for Trip)

    milestone "Done": event(of Book accommodation)
}
```

Fig. 1: Process for trip management modeled with DPIL

Orthogonal to the two modeling paradigms a process involves multiple *perspectives*. Activities in the process, for instance, follow a certain order (*functional* and *behavioral*), are performed by certain organizational resources (*organizational*) and require an appropriate handling of data (*data-oriented*) [2, 3]. Many DPMLs are limited to the functional and the behavioral perspectives. Even the more expressive *Dynamic Condition Response Graph* [4] language still lacks the involvement of organizational resources. In contrast, the *Declarative Process Intermediate Language (DPIL)* [3] has been designed for modeling process rules for *multiple* perspectives. Thus, using DPIL it is possible to model dependencies between activities as well as between resources and activities and all other combinations of elements from different or the same perspectives. DPIL and its expressiveness have been motivated and evaluated in both, industry applications like the KpPQ project[2] and in academical contexts [3] based on the well-known set of *Workflow Patterns*. One result of the academical investigation was that DPIL provides support for around 50% more of the requirements than BPMN does. However, though we decided to use DPIL as example language the principle can be applied to other DPMLs, too [5].

DPIL is a textual modeling language and allows for defining reusable rule templates (*macros*) in order to keep the resulting model as concise as possible. For instance, the *produces* macro (*produces(t,d)*) states that an activity t cannot be complete until a data object d has been produced. Consequently, the *consumes* macro (*consumes(u,d)*) stipulates that a second activity, u, cannot be started until this data object d is available. This results in a temporal ordering of the two activities t and u driven by a data dependency. Sometimes the temporal ordering of activities might depend on information which are not part of the process. Such constraints can be formulated using the *sequence* macro (*sequence(t,u)*), whereby u must be preceded by at least one execution of t. Resources can be assigned to an activity using the *role* macro (*role(t,r)*), which restricts an actor of t to be in role r. Such assignments do not need to be static, as the *binding* macro (*binding(t,u)*) shows. This macro says that the actor in the

[2] For more information we would like to refer to: `http://kppq.de/`

activities t and u has to be the same. The DPIL model in Fig. 1 is a simplified sketch of a business trip application process. It says that an application can be approved by the administration as soon as the application document is available. The accommodation has to be booked by the applicant and, at the earliest, as soon as the application has been approved. This step concludes the process.

Process model analysis and improvement phases can be supported by business process simulation techniques [6]. Simulation tools can produce log files which can be further analyzed enabling the user to predict the performance of a process in an operative context. As a second application, simulated logs can be used to gain a deeper understanding of the meaning and properties of a process. Cognitive research has shown that examples help to improve the comprehension of rules and abstractions [7]. A third advantage of simulation tools that produce logs is the opportunity to evaluate process mining techniques, as it has been shown in [8]. However, though there are many simulation techniques for *imperative* languages there is currently a lack of simulation tools for *declarative* languages. The approach in [8] is able to simulate declarative process models but the underlying DPML is limited to the functional and behavioral perspectives of processes. The transformation from multi-perspective declarative process models to an imperative representation is still vague. Consequently, it is vague, too, whether imperative simulation tools can be applied. We therefore complement the DPIL framework [9] with a multi-perspective declarative process simulation tool called *MuDePS* which is presented in the remaining part of the paper.

2 MuDePS: Overview and Demo Guidelines

In the demo we make use of two components of the DPIL framework, namely *DPIL Modeler* and *MuDePS* itself in order to demonstrate the simulation of the example process above as well as an extended version. The modeler ships with a textual editor based on the *Eclipse IDE* in combination with *Xtext*. Thus, the user is supported through general IDE features like auto-completion as well as through language features like syntax checking and reference resolution. MuDePS operates on models formulated using a logic language called *Alloy* [10]. The gap between a model specified in DPIL and its Alloy representation is closed using an automated model-to-text generator based on *Acceleo*.[3]

The MuDePS simulation tool works as follows. First, a DPIL model is translated into an Alloy-conform specification. Alloy ships with an analyzer which is able to identify *unique* examples for a given model and target size *exhaustively*. This means that each possible process execution path occurs exactly once in the resulting log which is serialized according to the *eXtensible Event Stream (XES)* standard format[4]. This standard is based on the idea of *Discrete-Event Simulation* which means that the log of a particular process execution is a chain of events whereby each encapsulates all relevant information about the executed activity and the executing resource and tools.

[3] Xtext: `www.eclipse.org/Xtext/`, Acceleo: `www.eclipse.org/acceleo`

[4] `http://www.xes-standard.org/`

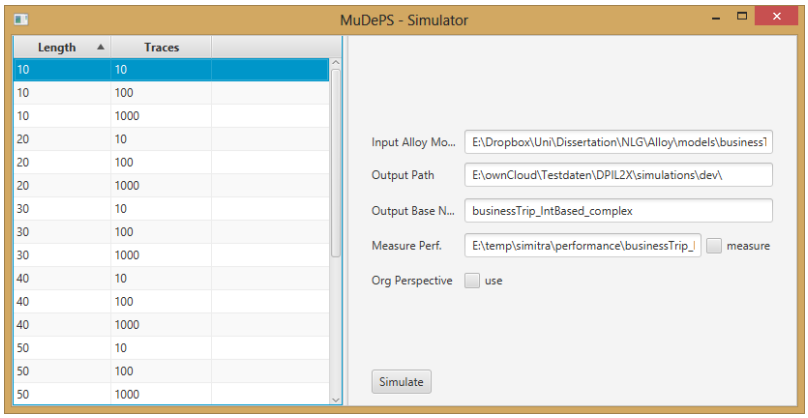


Fig. 2: MuDePS: User-interface of the simulation tool

A simulation model in MuDePS consists of two main parts: *(i)* A static part that describes the general structure and constraints for an intrinsically valid process event trace and *(ii)* a dynamic part consisting of constraints that have been generated from the source DPIL model. The static part ensures that each event occurs at a particular point in time (behavioral perspective) and that each event is able to encapsulate the activity name (functional perspective), the performer (organizational perspective) and data access information. An actor is usually part of an organizational structure. Thus, we also provide an Alloy implementation of the well known organizational meta-model discussed in [11].

The dynamic part contains the Alloy representation of the particular process model like the example in Fig. 1. The group, task and document elements are transformed to so called *signatures* which are comparable to *classes* in object-oriented programming languages. Thus, signatures can be abstract, too, and a signature that represents a process model element is an extension of one signature from the static part. A process rule is transformed to a so called *fact*. A fact is an invariant, i.e. a constraint that always must be fulfilled and that can be used to represent non-structural restrictions. In order to run the simulation we make use of Alloy's *run* command for which we have to provide a maximum target size of the solutions, i.e. the maximum process trace length. Running the command, the Alloy analyzer translates the Alloy model into a system of equations and applies a solver. MuDePS iterates over the resulting solutions considering the provided maximum trace length which, in turn, makes the solution space finite.

Based on its characteristics MuDePS can be used to produce an *exhaustive* set or a set of a *desired size* of example traces for a given DPIL model and maximum trace length. The traces are provided in the XES log format and can be post-processed by any desired application that operates on XES files.

3 Maturity and Future Work

As already shown in [9], the DPIL Framework is a well-evaluated prototype that is used for demonstration purposes in academic and industrial contexts.

The new part of the framework, MuDePS, shown in Fig. 2, has been evaluated regarding correctness and performance in [12] based on a real-life DPIL model. MuDePS itself, installation instructions as well as an example process and an exemplary generated event log are available at `http://mps.kppq.de`. Additionally we provide a screencast showing the basic workflow for simulating a DPIL model with MuDePS at: `https://youtu.be/JhqSiAxChKQ`. Currently the transformation from DPIL to Alloy is done using the signature of the macros. However, this means a restriction regarding DPIL's flexibility to create new process rule templates. That is why we are currently working on a more flexible transformation based on the actual rule structure. Additionally, the computation time for larger process event logs (> 80 events) is rather low (around 45 minutes for 1000 instances). However, this is caused by the multi-perspectivity but can be optimized, e.g. through partially inverting constraints in order to early minimize the solution space. Future versions of the tool will also consider activity life cycles (event start/completion) and modalities, i.e. soft and hard constraints. Additionally it will be fully integrated into the DPIL Modeler.

References

1. D. Fahland, D. Lübke, J. Mendling, H. Reijers, B. Weber, M. Weidlich, and S. Zugal, "Declarative versus imperative process modeling languages: The issue of understandability," in *BPMDS*, pp. 353–366, 2009.
2. R. Vaculín, R. Hull, T. Heath, C. Cochran, A. Nigam, and P. Sukaviriya, "Declarative business artifact centric modeling of decision and knowledge intensive business processes," in *EDOC*, pp. 151–160, 2011.
3. M. Zeising, S. Schönig, and S. Jablonski, "Towards a Common Platform for the Support of Routine and Agile Business Processes," in *CollaborateCom*, 2014.
4. T. T. Hildebrandt and R. R. Mukkamala, "Declarative event-based workflow as distributed dynamic condition response graphs," *PLACES 2010 (Journal v.)*, 2011.
5. L. Ackermann, S. Schönig, and S. Jablonski, "Simulation of multi-perspective declarative process models," in *Int. Conf. on BPM*, Springer, 2016.
6. W. M. P. van der Aalst, "Business Process Simulation Revisited," *Enterprise and Organizational Modeling and Simulation*, vol. 63, pp. 1–14, 2010.
7. A. L. Brown and M. J. Kane, "Preschool children can learn to transfer: Learning to learn and learning from example," *Cogn. Psychology*, vol. 20, pp. 493–523, 1988.
8. C. Di Ciccio, M. L. Bernardi, M. Cimitile, and F. M. Maggi, "Generating event logs through the simulation of declare models," in *EOMAS*, pp. 20–36, 2015.
9. S. Schönig and M. Zeising, "The DPIL framework: Tool support for agile and resource-aware business processes," in *BPM Demo Session*, pp. 125–129, 2015.
10. D. Jackson, *Software Abstractions: logic, language, and analysis*. MIT press, 2012.
11. C. Bussler, "Analysis of the organization modeling capability of workflow-management-systems," in *PRIISM*, pp. 438–455, 1996.
12. L. Ackermann, S. Schönig, and S. Jablonski, "Simulation of Multi-Perspective Declarative Process Models," *preprint: https://epub.uni-bayreuth.de/id/eprint/2881*, 2016.

ResRec: A Multi-criteria Tool for Resource Recommendation

Michael Arias, Eric Rojas, Jonathan Lee, Jorge Munoz-Gama, and Marcos Sepúlveda

Computer Science Department, School of Engineering
Pontificia Universidad Católica de Chile, Santiago, Chile
{m.arias,eric.rojas,wllee}@uc.cl, {jmun,marcos}@ing.puc.cl

Abstract. Dynamic resource allocation is considered a key aspect within business process management. Selecting the most suitable resources is a challenge for those in charge of making the allocation, because the efficiency with which this task is executed, can contribute to the quality of the results, and improve the process performance. Different mechanisms have been proposed to improve resource allocation. However, there is a need for more flexible allocation methods that integrate a set of conditions and requirements defined at run-time, and also, allow the combination of different criteria to evaluate resources. In this paper, we present ResRec, a novel Multi-factor Criteria tool that can be used to recommend and allocate resources dynamically. The tool provides the feature of solving individual requests (On-demand), or requests made in blocks (Batch) through a recommender system developed in ProM.

Keywords: resource allocation, process mining, business processes, recommendation systems, organizational perspective

1 Significance of Resource Recommendation for BPM

The increasing interest in the study of resources and their relationship within the organizational mining perspective [1] has helped to generate insight on how to improve the process efficiency. One of the main categories within this perspective is the resource allocation. The existing allocation approaches aim to provide mechanisms to coordinate in a better way the resources involved in their business processes. Allocate resources dynamically is a relevant challenge [7]. It implies the selection of the most appropriate resource to execute a request created at run-time, considering a set of specific process characteristics that influence the decision of what resource allocate to execute the corresponding activities. Few methods pay attention to the information contained in event logs and organizational models to allocate resources dynamically. For instance, Reinforcement learning [5] has been used to adjust and optimize the resource allocation based on learning appropriate policies incorporating a feedback analysis at run-time;

whereas, a Naïve Bayes Model approach has been proposed to select the best performer to execute an activity based on the total completion time [6]. There is a need to optimize the task of allocating resources through methods that: i) support resource allocation at run-time; ii) use different criteria to evaluate resources, combining process information with other additional information (e.g., resource characteristics); iii) execute the allocation considering different abstraction levels (e.g., activity, sub-process, or process level); and iv) support different decision strategies faced by the person in charge of making the allocation in their daily processes. In this paper, we present ResRec, a novel multi-criteria tool to allocate/recommend resources dynamically. The tool combines different criteria to assess resources, allows the definition at run-time of individual requests (On-demand) or requests made in blocks (Batch), and provides distinct decision alternatives –a decision maker-oriented approach–. The remainder of the article is structured as follows. Section 2 introduces concepts related with the ResRec framework for resource allocation/recommendation. In Section 3 we present the details about the ResRec implementation. Section 4 presents a demonstration of the tool and its functionality. Finally, Section 5 presents the work conclusions.

2 The Resource Recommendation Framework

The proposed framework for allocating/recommending the most suitable resources for executing activities in a business process is shown in Figure 1. For allocation we mean selecting the most appropriate resource. For recommendation we mean retrieving a ranking of the most suitable resources that could be considered by the person in charge of the allocation. The user responsible for undertaking the allocation specifies the characterization of a resource allocation/recommendation. This characterization is integrated by different factors that represent the desired request properties, e.g., what part of the process is the resource request for or what specific characteristic the process instance has. For example, in a Help-Desk process, $Contact\ level\ 1 - English$ are two factors that can characterize a request. Resources are evaluated considering several criteria: frequency, performance, quality, cost, expertise and workload. Furthermore, specific weights are defined according to the importance level that the responsible wants to give to each criterion.

We use an information repository to store the resource contextual data (e.g., expertise), and the historical information about past process execution (e.g., frequency, performance, quality, and cost). We created a knowledge base, where we use a Resource Process Cube to abstract at a conceptual level (not at an implementation level) the historical information, and expertise matrices to represent the expertise of a resource and the desired level of expertise required to execute a given characterization. Each available resource to execute a given characterization is evaluated considering different criteria, which are computed across several metrics defined in [4]. The definition of the Resource Process Cube is closer to the well-known OLAP cubes, providing slice and dice operations for the analysis of each specific characterization and resource. We propose

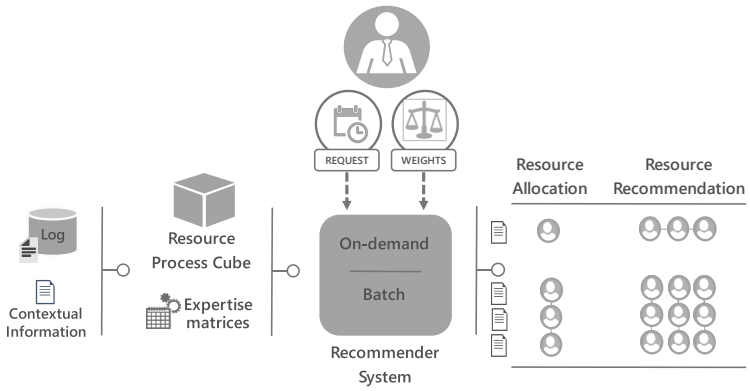


Fig. 1. General overview of the framework to allocate/recommend resources

four use cases: 1) On-demand Resource Allocation: a resource is allocated to a single request; 2) On-demand Resource Recommendation: a ranked list of resources is recommended for a single request; 3) Batch Resource Allocation: a batch of requests is evaluated, and a resource is allocated to each of them; and 4) Batch Resource Recommendation: a batch of requests is considered, and a ranked list of resource tuples is recommended to perform all requests. We use a recommender system that uses two methods to allocate/recommend the most suitable resources: a method based on Integer Linear Programming (to support On-demand or Batch Resource Allocation), and a method based on the Best Position Algoritm (BPA2) [2] (to support On-demand or Batch Resource Recommendation). The details about the proposed methods and the implementation will be published shortly [3].

3 ResRec: The Tool

To perform a dynamic resource allocation/recommendation, the ResRec tool provides two plugins implemented in ProM: 1) *GenerateResourceKnowledge*, and 2) *Recommend Resources*. Both plugins are available in the *Resource Recommendation* package in the open-source framework ProM (available in ProM nightly-builds [2]).

3.1 Generate Resource Knowledge

To use the ResRec tool, we have first to import the information (contextual and historical) needed for the decision making within the framework (Figure 2a). For this purpose, we created a standardized format to store the information that is

[2] www.promtools.org/prom6/nightly

considered. ResRec includes an extension to the Java OpenXES library [3] that allows to standardize and manipulate the information that is used to generate the required knowledge. Using the *GenerateResourceKnowledge* plugin (Figure 2b), the different criteria are evaluated through different metrics, generating the resource knowledge.

3.2 Recommend Resources

Based on the resource knowledge, we apply the *RecommendResources* plugin (Figure 2c). This plugin allows the configuration of the required parameters to obtain the final allocation/recommendation. Figure 2d shows the needed configuration. First, the person in charge of making the allocation must decide which method wants to perform: resource allocation (single resource), or resource recommendation (a ranking of resources). Then, the person in charge specifies the amount of requests for each type of available characterizations (On-demand for 1 request, Batch for 2 or more requests). After that, weights are defined to describe the importance of each criterion considered in the framework. Finally, the recommender system computes the allocation/recommendation (Figure 2e) according with the chosen configuration, showing as an output the resource(s) allocated/recommended for each request.
Note that the ResRec tool is designed to be generic and extensible, being able to incorporate new criteria to evaluate the resources, so as to make the resource allocation a more effective task. Moreover, the framework can be adapted to be used in any domain or scenario.

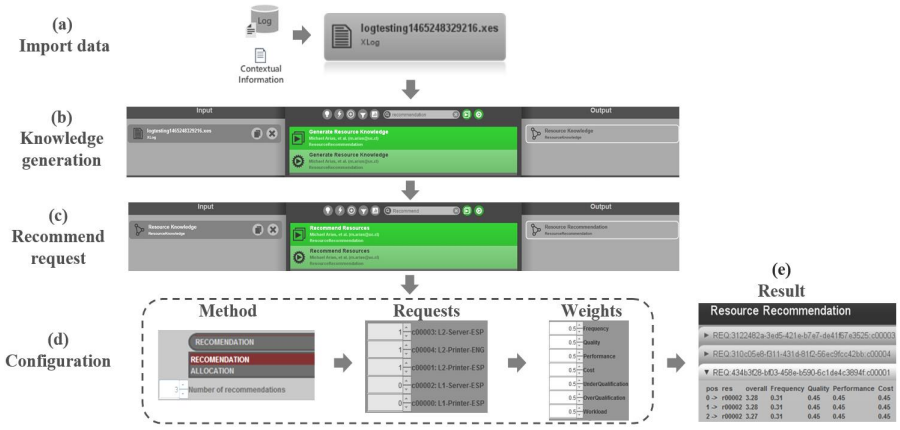


Fig. 2. General walkthrough of the ResRec tool in ProM

[3] https://svn.win.tue.nl/trac/prom/browser/Packages/ResourceRecommendation/Trunk/src/org/processmining/resourcerecommendation/utils/resrecxes

4 Demonstration

A screencast that demonstrates the usage of the ResRec tool is available on the web (http://is.ing.uc.cl/dcc/index.php/resrec/). The screencast shows the steps followed to generate the resource allocation/recommendation considering the use cases introduced in Section 2. We used a Help-Desk process as a running example. First, we add the contextual and the historical information required to evaluate the resources. Then, we create the knowledge base needed to perform the resource recommendation. After that, we configure the corresponding parameters in order to generate a Batch Resource Recommendation, defining a batch of requests and the weights describing the importance of each criterion. We show the obtained results by applying the method based on BPA2, performing a Batch Resource Recommendation. Additional to the Batch Resource Recommendation, the other three use cases outlined are also presented. Furthermore, we used the ResRec tool to recommend resources in a real life scenario. The case study included the resource allocation/recommendation for a consulting firm specializing in the sale of business software solutions in Costa Rica, based on its help-desk process.

5 Conclusions

We presented ResRec, a multi-criteria tool that consider historical and contextual information to allocate/recommend the most suitable resources to execute either a single request or a batch of requests, defined at run-time. To obtain the final recommendation, we propose a recommender system that use two methods: 1) Resource allocation based on Integer Linear Programming, and 2) Resource recommendation based on the Best Position Algorithm (BPA2). This tool also has been tested through a real-life help desk scenario of a software consulting company that handles a management solution developed for the automotive industry named DMS-One SAP system. We considered a event log with 1.778 cases registered between August and November 2015. As a future work, we may consider integrating a greedy-based approach as an alternative method to produce a Batch Resource Recommendation with several requests that need to be resolved simultaneously. We also plan to evaluate the incorporation of new criteria to enrich the knowledge base available to determine the most suitable resources. Finally, we aim to apply the framework to allocate/recommend resources with more case studies.

Acknowledgments. This project was partially supported by the Ph.D. Scholarship Program of CONICYT Chile (CONICYT-Doctorado Nacional 2014 - 63140181), Universidad de Costa Rica Professor Fellowships, and by Fondecyt (Chile) Project No.1150365.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)

2. Akbarinia, R., Pacitti, E., Valduriez, P.: Best position algorithms for efficient top-k query processing. Inf. Syst. 36(6), 973–989 (2011)
3. Arias, M., Munoz-Gama, J., Sepúlveda, M., Carmona, J., Miranda, J.C.: On-demand and batch resource allocation/recommendation based on multi-factor criteria. (under revision) (2016)
4. Arias, M., Rojas, E., Munoz-Gama, J., Sepúlveda, M.: A framework for recommending resource allocation based on process mining. In: BPM Workshops (2015)
5. Huang, Z., van der Aalst, W.M.P., Lu, X., Duan, H.: Reinforcement learning based resource allocation in business process management. DKE 70(1), 127–145 (2011)
6. Wibisono, A., Nisafani, A.S., Bae, H., Park, Y.: On-the-fly performance-aware human resource allocation in the business process management systems environment using naïve bayes. In: AP-BPM 2015. pp. 70–80 (2015)
7. Zhao, W., Zhao, X.: Process mining from the organizational perspective. In: Foundations of Intelligent Systems, pp. 701–708 (2014)

PTandLogGenerator: a Generator for Artificial Event Data

Toon Jouck and Benoît Depaire

Hasselt University, Agoralaan Bldg D, 3590 Diepenbeek, Belgium
toon.jouck@uhasselt.be; benoit.depaire@uhasselt.be

Abstract. The empirical analysis of process discovery algorithms has recently gained more attention. An important step within such an analysis is the acquisition of the appropriate test event data, i.e. event logs and reference models. This requires an implemented framework that supports the random and automated generation of event data based on user specifications. This paper presents a tool for generating artificial process trees and event logs that can be used to study and compare the empirical workings of process discovery algorithms. It extends current tools by giving users full control over an extensive set of process control-flow constructs included in the final models and event logs. Additionally, it is integrated within the ProM framework that offers a plethora of process discovery algorithms and evaluation metrics which are required during empirical analysis.

Keywords: artificial event logs; process simulation; process discovery

1 Generating Event Data for Algorithm Evaluation

Process discovery techniques are concerned with discovering the control-flow of a process directly from an event log. During the last decade many process discovery techniques have been developed (see [2] for an overview). Currently new techniques are developed to outperform others in term of model quality measures. This has led to an increasing importance of evaluating and comparing existing algorithms empirically [2]. In order to perform such an evaluation, a set of appropriate test event data, i.e. event logs and reference models, is required.

Generally, three requirements with regard to test data must hold while doing empirical analysis of process discovery algorithms. Firstly, a researcher should have full control over the control-flow characteristics of the event data generated. A second requirement is randomness to prevent wrong generalizations based on non-random event data. Finally, the final event logs and reference models should be in the standard format[1] to ensure their compatibility with tools that implement process discovery algorithms and evaluation metrics.

[1] Event logs should conform to the XES standard: http://www.xes-standard.org

The presented tool *PTandLogGenerator* fulfills all the requirements stated above as it enables the random and automated generation of process trees and event logs based on user-defined control-flow specifications. It applies a generic two-step approach: generate a process tree, then simulate this tree into an event log described in [4]. Firstly, the user specifies the control-flow constructs, defined as meaningful process blocks that will be included in the generated process trees. In the second step, the trees are simulated into event logs.

The idea of implementing an artificial data generator is not new. However, the existing tools still have some limitations with regard to the requirements stated above. The most advanced tool, PLG2 [1], allows for control of the basic workflow patterns, but does not allow for more complex constructs such as duplicate activity labels or long-term dependencies. Moreover, PLG2 is not directly integrated within the ProM framework, which is a disadvantage when doing empirical process discovery evaluation.

The remainder of this paper describes how to use the *PTandLogGenerator* tool in the case of comparing two process discovery algorithms.

2 Walkthrough of the Process Tree and Log Generator

The *PTandLogGenerator* tool is available as a package in the open-source framework ProM[2]. This section will describe the different steps of generating a sample of event logs needed to evaluate two process discovery algorithms. Consider the comparison of the α++ miner [6] and the Inductive Miner [5] on logs including long-term dependencies, i.e. causal dependencies between tasks in different exclusive choice constructs. Consequently, one needs a set of event logs containing such long-term dependencies, while controlling for other control-flow constructs. The following paragraphs show how these can be created using the *PTandLogGenerator*, see `https://drive.google.com/file/d/0B9nT4OtWjscVOV94VDEwb3I4V2s/view?usp=sharing` for a screencast.

A Population of Process Trees The starting point is the definition of a process tree population. These process trees contain a combination of control-flow constructs (CFC), i.e. process tree building blocks, that are used as population parameters to describe the population. The user can assign probabilities to each of the CFC to express the probability that these constructs are added to a tree within the population. We distinguish between three types of constructs: activities, workflow patterns and complex constructs:

- Activities (see area 1 in Fig. 1): users can influence the size of the trees included in the population by specifying the triangular distribution of the number of activities by assigning a minimum, a mode and a maximum. Each time a process tree is generated, i.e. drawn from the population, a random number for the number of activities is taken from that triangular distribution.

[2] Available in the ProM nightly builds at `http://www.promtools.org/`

– Workflow control-flow patterns (see area 2 in Fig. 1): basic fundamental patterns common to all business processes. These patterns include sequence, exclusive choice, multi-choice, concurrent behavior and loops, represented by the following operator nodes in Process trees: $\rightarrow, \times, \vee, \wedge$ and $\circlearrowleft$.
– Complex constructs (see area 3 in Fig. 1): more complex control-flow constructs include silent activities, reoccurring activities (i.e. duplicate labels), long-term dependencies and infrequent paths. The last construct assigns unequal branch probabilities to each of the outgoing branches of an exclusive choice in order to make some paths less frequent in the process.

In our example use case, we want to evaluate two algorithms on long-term dependencies. Therefore we define a population of process trees with 50% probability of inserting long-term probabilities. The definition of this population can be configured using the settings wizard shown in Fig. 1.

Generating a Random Sample of Trees The next step involves drawing a random sample, i.e. generating random process trees, from the previously specified population. The size of this sample can be specified in the tree generator settings as can be seen in area 4 of Fig. 1. For the example use case we generate a random sample of size 10. The algorithm described in [4] is implemented to build each process tree in a stepwise manner. It uses the probabilities specified in the population as input parameters to randomly add nodes to the tree.

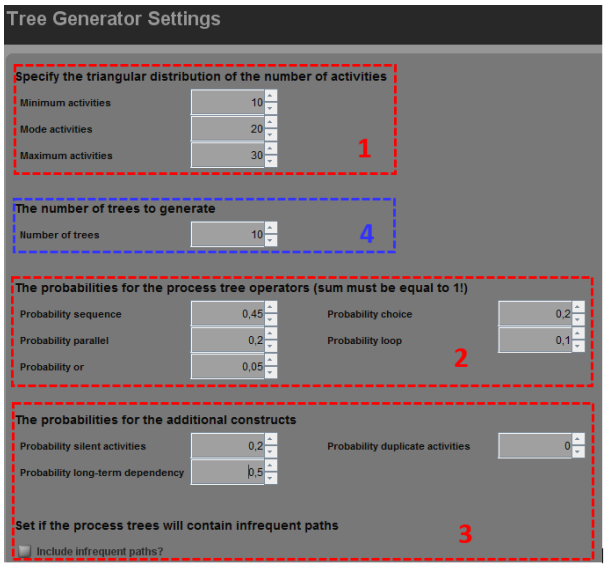


Fig. 1: The Population of Process Trees in the Example

The output of this step is a set of process trees in the standard PTML-format. In this way the complete toolbox for import/export, analysis and visualization of process trees integrated into ProM is available to the user. The screenshot in Fig. 2 shows the visualization pane for tree number 3 in the sample. The annotations on the branches represent their execution probabilities. The tree shown in Fig. 2 contains a long-term dependency between the activity h and c expressing the causal relationship that if h is executed, c can never follow later on. For further

information, the reader is referred to the tree generating algorithm described in [4].

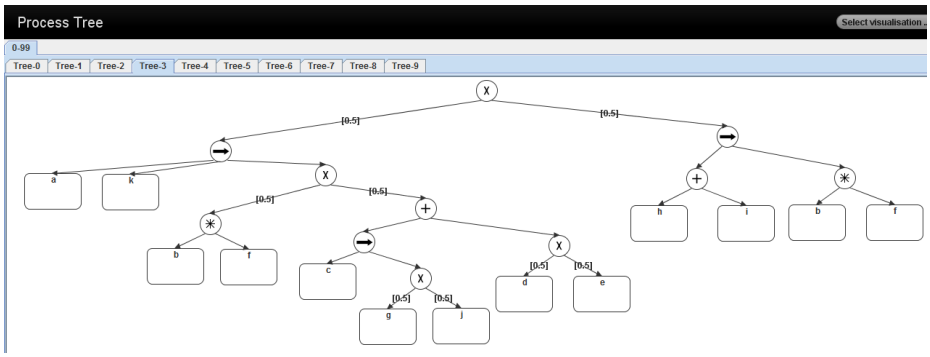
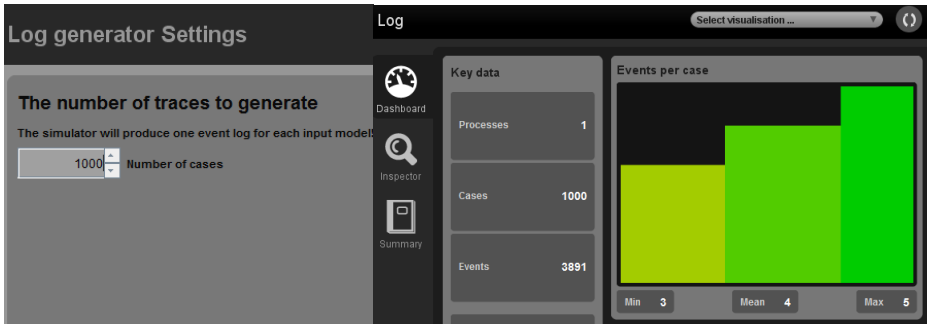


Fig. 2: A process tree from the random sample visualized

Simulating Trees Into Event Logs Then in the third step the tool enables users to generate (an) event log(s) for each process tree in the sample. Each process tree can be seen as a population of event logs: an event log is a multiset of traces simulated from that process tree. The tool allows users to specify the number of traces that the final event log(s) will contain as shown in Fig. 3a. In the example case we generate one event log with 1000 traces for each tree. The resulting event logs are in the standard XES-format providing all the log functionalities provided in ProM. Fig. 3b shows the log view for the event log generated from the tree in Fig. 2.



(a) The Number of Traces in the Generated Event Logs

(b) Log View of the Generated Event Log

Fig. 3: Log Generator

Evaluating Process Discovery Techniques Once the event logs are generated, the empirical analysis of the evaluation of process discovery technique

is enabled. In the running case we apply the α++ miner [6] and the Inductive Miner [5] in ProM and calculate quality metrics for each discovered model. In the running case the discovered models of the inductive miner have an average fitness value of 100% and a precision value of 58.5%, whereas the models discovered by α++ miner have a lower average fitness value of 32.1% and a higher average precision value of 77.9%. These results are used to demonstrate the possible use cases of the tool. A more thorough empirical analysis would need more observations which is outside the scope of this paper.

3 Maturity and Use Cases

The tool has reached a high level of maturity which enables its use in large scientific experiments. This has been proven by the successful application of the tool in the large scale empirical assessments in [3]. Furthermore, the organizers of the first process discovery contest[3] have chosen this tool to create the benchmark event logs as it allows users the full control over an extensive range of control-flow constructs.

Acknowledgements

Special thanks to Alfredo Bolt and dr. Massimiliano de Leoni for their help on implementing and improving this tool.

References

1. Burattin, A.: PLG2: Multiperspective Processes Randomization and Simulation for Online and Offline Settings. ArXiv e-prints (1506.08415) (Jun 2015)
2. De Weerdt, J., De Backer, M., Vanthienen, J., Baesens, B.: A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. Information Systems 37(7), 654–676 (Nov 2012), http://www.sciencedirect.com/science/article/pii/S0306437912000464
3. Janssenswillen, G., Jouck, T., Creemers, M., Depaire, B.: Measuring the quality of models with respect to the underlying system: An empirical study. In: Business Process Management 2016. Springer (accepted)
4. Jouck, T., Depaire, B.: Generating Artificial Data for Empirical Analysis of Process Discovery Algorithms: a Process Tree and Log Generator. Technical Report, Universiteit Hasselt, Universiteit Hasselt (Mar 2016), http://hdl.handle.net/1942/20818
5. Leemans, S.J., Fahland, D., van der Aalst, W.M.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Business Process Management Workshops. pp. 66–78. Springer (2014), http://link.springer.com/chapter/10.1007/978-3-319-06257-0_6
6. Wen, L., van der Aalst, W.M., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Mining and Knowledge Discovery 15(2), 145–180 (2007), http://link.springer.com/article/10.1007/s10618-007-0065-y

[3] http://www.win.tue.nl/ieeetfpm/doku.php?id=shared:process_discovery_contest

Enabling Batch Processing in BPMN Processes

Luise Pufahl and Mathias Weske

Hasso Plattner Institute at the University of Potsdam, Germany
{Luise.Pufahl,Mathias.Weske}@hpi.uni-potsdam.de

Abstract. Business process automation improves organizations' efficiency to perform work. Single executions of process models, called process instances, are usually executed independently in business process management systems (BPMS). In practice, we can observe examples in which a synchronized execution of groups of instances for certain activities, called batch processing, can lead to an improved performance. Batch regions is a concept to allow batch processing in business processes. This demo presents the implementation of the batch region concept in an open-source BPMN engine. It shows how, with a few extensions only, batch processing is enabled and how the consolidated view of several work items in one user form, leads to an improved work efficiency for users.

Keywords: Batch Processing, Process Enactment, BPMN

1 Introduction

Business process automation improves organizations' efficiency to perform work. Therefore, processes are often captured in process models, typically in BPMN (Business Process Modeling and Notation) [2] diagrams, the industry standard. These are then executed by a BPMS (e.g., Bizagi [1], Camunda [3], Signavio Workflow [8]). An executing instance of a process model is called process instance. Multiple process instances might run simultaneously in a BPMS. However, as Russell et al. [7] observe, "each of these is assumed to have an independent existence and they typically execute without reference to each other."

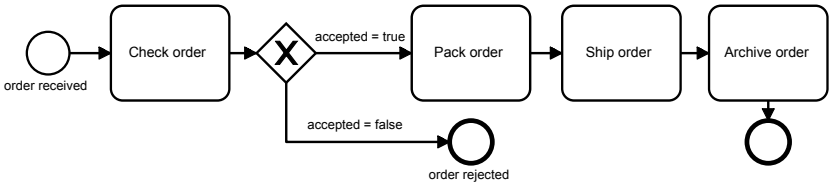


Fig. 1. Simplified online retailer process shown in a BPMN diagram.

In practice, we can observe different cases where the synchronized execution of several instances is beneficial and can improve process performance. For example, Fig. 1 shows the simplified version of an online retailer process as BPMN diagram in which

customer orders are handled. Often, online retailers do not charge any transport cost with the effect that customers place multiple orders in relatively short time frames. In such situation, several orders of the same customer could be packed and shipped together to save shipment costs. This approach, called batch processing, allows business processes which usually act on a single item, to bundle the execution of a group of process instances for certain activities in order to improve performance. Other examples which benefit from batch processing, we can observe in health care, e.g. collecting a set of blood samples for delivery to the laboratory, in insurance and finance, e.g. consolidating several letters to one customer and send them as one mail, and in administration, e.g. collecting several invoices for their approval. Usually these examples are already executed in a batch, but manually with the risk that batch processing rules might not be clear for everyone or might be ignored. Recent research approaches [4–6] provide means to integrate batch processing in business processes models and its automatic execution. In contrast to the others, the batch region concept in [6] allows an individual batch configuration based on which instances are batched with similar data characteristics over a number of activities. In this demo, we want to show an implementation of the batch region concept for BPMN processes in the open-source BPM platform Camunda [3]. Next, Section 2 introduce the batch region concept for BPMN processes; Section 3 presents the implementation details. We conclude in Section 4.

2 Batch Region Concept in BPMN - Design and Execution

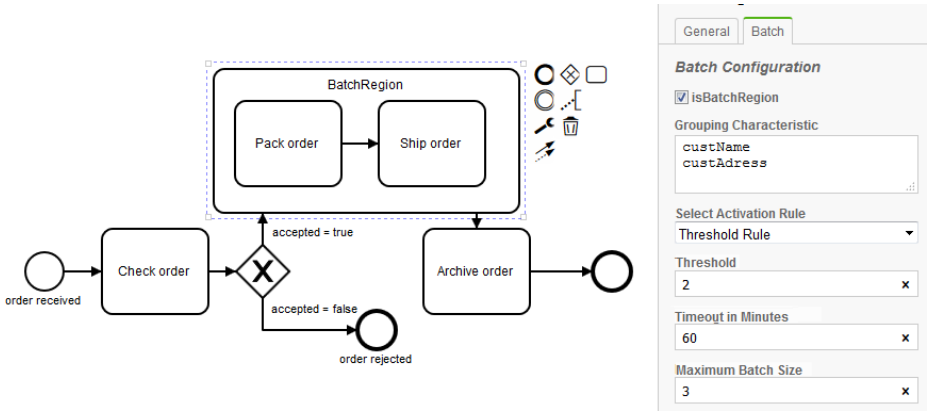


Fig. 2. Online retailer process with a batch region to save shipping costs.

A batch region in a BPMN process diagram is a special type of sub-process enabling batch processing for its activities. Fig. 2 shows the online retailer process with a batch region surrounding the activities *Pack order* and *Ship order*. With its configuration parameters (visualized in the right panel of Fig. 2), the process designer is able to specify the conditions for the batch execution. Those are (1) a grouping characteristic to cluster process instances to be processed in one batch based on data attributes (e.g., the `custName` and `custAdress` to identify similar order instances in our retailer example), (2) an activation rule to determine when a batch is activated while balancing

between waiting time and costs savings (e.g., when at least two similar order instances are available or a timeout of one hour is reached specified in a threshold rule), and (3) the maximum batch size indicating the maximum number of instances in a batch (e.g., at maximum three orders fit in one parcel). In a batch region, XOR gateways are not allowed because decisions are usually taken on individual items, but not on a item group. Further, we require that only one start event in a batch region so that the activation of a batch cluster is uniquely determined.

Each execution of the batch region is represented by a batch cluster. It collects a number of sub-process instances for batch execution whereby these are assigned based on their data values. For example, only instances with $\texttt{custName} = Johnn$ and $\texttt{custAddress} = Madrid$ are assigned to the cluster `John_Madrid`. A batch cluster has different life cycle states shown in Fig. 3. When the first batch region activity is enabled, a sub-process instance gets assigned to an existing or new batch cluster. It is checked whether a cluster is available with the same data characteristics that is in the *init* or *ready* state. If not, a new cluster is created in the initial state *init*. If the activation rule is fulfilled, it transitions into the *ready* state.In this state, a *batch work item* including data of all instances is provided to the task performer. In the *ready* state, further instances can be still assigned to it to achieve an optimal cluster utilization. In this state, the cluster can transition into the *maxloaded* state, if its maximum batch size is reached, or into the *running* state, if the task performer begins the work item execution. In these two states, no further instances can be added. The cluster stays in the *running* state for all further activities in the sub-process. With termination of the last batch work item, it changes into the *terminated* state. Now, the instances are again independent from each other. After this short introduction into batch region design and its execution semantics, the next section describes our extensions of an open-source BPMN engine to enable batch processing.

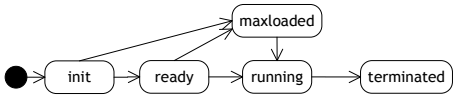


Fig. 3. Life cycle of a batch cluster

3 Tool Architecture and Implementation

For the implementation[2] of the presented batch region concept, Camunda [3] was selected, a Java-based, open source engine specifically tailored for a subset of BPMN.

First, we integrated batch region concept in the BPMN XML specification by utilizing *extension elements*, which the BPMN specification [2] explicitly supports to add new attributes and properties to existing constructs. The extension was added to the sub-process element describing the batch region configuration. Based on it, the Camunda modeler *bpmn.io* was adapted to enable a quick design of batch regions. Specifically, the sub-process element was extended as shown in Fig. 1.

The Camunda engine was extended by only four additional classes: The *batch region* class stores the configurations of each identified batch region and manages the assignment of process instances to a batch cluster. The *batch cluster* class governs the batch execution for its assigned group of process instances. The *batch behavior* class

[2] A link to repository of the implementation and a screen cast are available at `http://bpt.hpi.uni-potsdam.de/Public/BatchProcessing`.

includes the internal behavior of the activities in a batch region sub-process. In the Camunda engine, every process activity gets a task behavior (e.g., user task, service task) assigned, describing the internal activity behavior. In order to reuse these behaviors and limit the engine extension, the *batch behavior* inherits the normal task behavior and has additional methods for the batch execution defined in an interface. This is driven by the idea that one of the cluster instances leads the batch execution by first merging the data of all cluster instances and then executing the usual task behavior. Currently, this is implemented only for user tasks, but can easily applied to other task types. Further, a *batch timer* job class was added to enable the time-out defined in the threshold rule. Additionally, the BPMN parser was adapted to read batch regions' specifications and to assign every batch region activity its batch behavior.

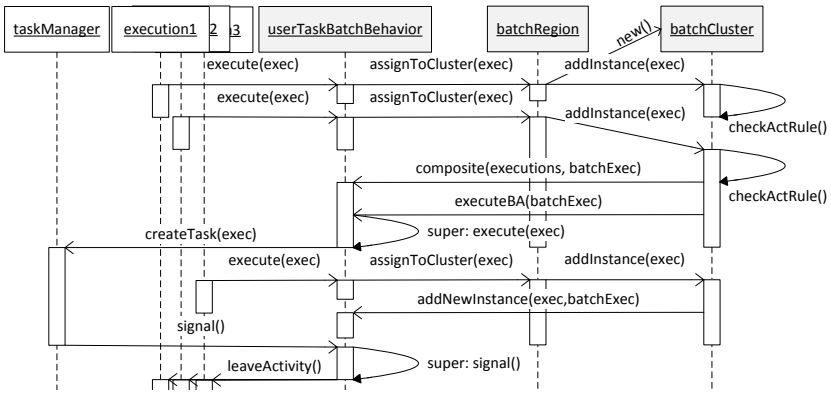


Fig. 4. Sequence diagram visualizing the interaction between added classes *batch behavior*, *batch region*, and *batch cluster* to the Camunda engine.

For the example of the *UserTaskBatchBehavior*, the interaction of the batch behavior with the batch region and the batch cluster class is shown in Fig. 4. As soon as an *Execution* object representing a process instance enables an activity with a batch behavior, it is added by the batch region to a cluster. If no batch cluster is currently available, it is first created and then the `add()`-method of the cluster is called in which also the activation rule is checked. Currently, our implementation supports the threshold rule. With its fulfillment, the cluster calls the `composite()`-method of the batch behavior merging the data of all instances. In case of the user task, a JSON variable with all instance data is created. This can be later reused during the user form design. Then, the `executeBA()`-method is called in which the batch behavior calls the `execute()`-method of its super class. Now, the normal user task behavior is executed in which a work item for the task performer is created. Fig. 5 shows the batch work item for the *Ship order* activity of the retailer example. We have used the JSON variable to visualize all orders in a table. The task performer can easily inspect all orders and has to enter the value for the logistics provider only once, as it is valid for all orders. Instead of three work items, the task performer has to process only one, leading to time and cost savings.

Our implementation provides also the feature to add new instances, while the first batch region activity is not completed, yet. The corresponding `addNewInstances()`-method simply adapts the JSON variable. With completion of a batch work item, the

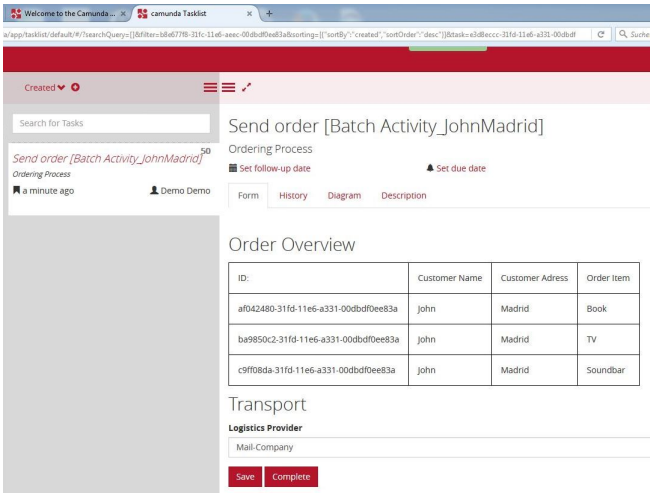


Fig. 5. Batch work item for the *Ship order* activity of the online retailer example.

task manager calls the `signal()`-method of the batch behavior distributing new added data (e.g., the logistics provider in Fig. 5) to all other cluster instances. Finally, with the last batch work item, also the batch cluster is terminated. The implementation shows that only small extensions on a BPMS are necessary, which also have no significant influence on the engine performance, to enable batch processing.

4 Conclusion

Real-world examples show the necessity of batch processing in business processes. In this paper, the batch region concept was implemented in an existing BPMS to realize batch processing for BPMN processes. Only small extensions are necessary to adapt the BPMN engine having also no significant impact on the engine performance. The demo shows that batch processing has the advantage that task performers can handle several items consolidated in one user form improving their work efficiency.

References

1. Bizagi: Bizagi Business Platform. `http://www.bizagi.com/`
2. OMG: Business Process Model and Notation (BPMN), Version 2.0 (2011)
3. Camunda: Camunda open-source BPM Platform. `https://www.camunda.org/`
4. Liu, J., & Hu, J.: Dynamic batch processing in workflows: Model and implementation. In: Future Generation Computer Systems, 23(3), pp. 338-347. Elsevier (2007).
5. Natschläger, C., Bögl, A., Geist, V., & Biró, M.: Optimizing Resource Utilization by Combining Activities Across Process Instances. In: Systems, Software and Services Process Improvement, pp. 155-167. Springer (2015).
6. Pufahl, L., Meyer, A., and Weske, M.: Batch regions: process instance synchronization based on data. In: EDOC, 2014 IEEE 18th International, pp. 150-159. IEEE, (2014).

7. Russell, N., Ter Hofstede, A. H., Edmond, D., & van der Aalst, W. M.: Workflow data patterns: Identification, representation and tool support. In: Conceptual Modeling-ER 2005. LNCS, vol. 3716, pp. 353-368. Springer (2005).
8. Signavio Workflow `https://workflow.signavio.com/`

Behavior-Based Process Comparison in Apromore

Abel Armas-Cervantes[1], Nick R.T.P. van Beest[2], Marlon Dumas[3], Luciano García-Bañuelos[3], and Marcello La Rosa[1]

[1]Queensland University of Technology, Australia
{abel.armascervantes,m.larosa}@qut.edu.au
[2]Data61, CSIRO, Australia
nick.vanbeest@data61.csiro.au
[3]University of Tartu, Estonia
{marlon.dumas,luciano.garcia}@ut.ee

Abstract. This paper presents the integration of three behavior-based comparison operations between logs and/or models into the Apromore process model repository. Each of these operations takes as input a pair of process artifacts (two event logs, two models, or one log and one model) and describes their behavioral differences by means of natural language statements. The generated difference diagnosis has a range of applications of interest to both practitioners and researchers. For example, the difference diagnosis can offer guidance for reconciling discrepancies between business process variants (*model2model* comparison); can be used to pinpoint and explain differences between actual and expected behavior for conformance checking purposes (*model2log* comparison); or can explain dissimilarities between normal and deviant executions of a process (*log2log* comparison).

Keywords: Apromore, behavioral comparison, deviance mining, conformance checking, consolidation of variants

1 Overview

The behavioral comparison of process models and event logs is a recurrent primitive in business process analysis. Conformance checking, deviance mining and (behavior-based) process model comparison can be seen as specific instances of this general primitive. In particular, conformance checking aims at finding if the observed behavior captured in an event log complies with the behavior specified in the model; deviance mining compares the behavior captured in event logs to explain why a business process deviates from its normal or expected behavior; finally, (behavior-based) process model comparison aims at explaining the behavioral differences between process models that may correspond to variants of the same business process.

A critical feature of behavioral comparison operations is the interpretability of the identified differences between logs, models or between a model and a log. In this regard, a set of techniques for conformance checking, deviance mining and process model

comparison have been proposed in [1, 2], [3] and [4, 5], respectively. These techniques use event structures [6], a well-known model of concurrency, as unified behavioral abstractions for both logs and models [7]. This comparison approach is *independent of the input-format*, i.e. process models, event logs or both. The differences are reported to the user in the same way regardless of the input type, using a set of sentences in natural language, without requiring in-depth knowledge of formal modeling languages like Petri nets. Consequently, our tool is suitable for use by business analysts and, as opposed to other approaches, it does not require process modeling or mining experts to interpret the results. Figure 1 depicts an overview of the implemented operations.

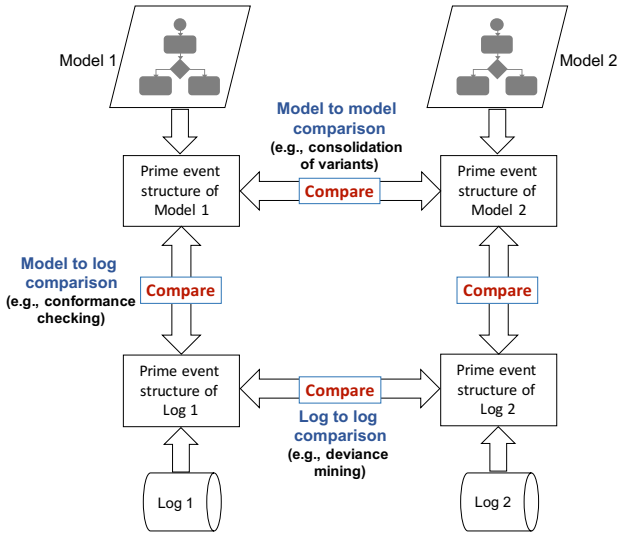


Fig. 1: Overview of the implemented comparison operations.

This paper presents the integration of the techniques proposed in [1, 2], [3] and [4, 5] into the Apromore process model repository [2]. Apromore is an open-source and extensible online repository of state-of-the-art capabilities for managing large process model collections. The operations for conformance checking, deviance mining and (behavior-based) process model comparison complement the wide range of existing capabilities, e.g., process model merging, simulation, restructuring, querying and similarity search, provided by Apromore. The three techniques are wrapped into a pairwise *Compare* operation that will automatically execute the appropriate comparison operation depending on the type of selected artifacts as input, two models, one model and one log, two logs. The modeling languages supported are all those available in Apromore, namely BPMN, EPCs, Petri nets and YAWL. The log format can be XES or MXML.

[2] http://apromore.qut.edu.au

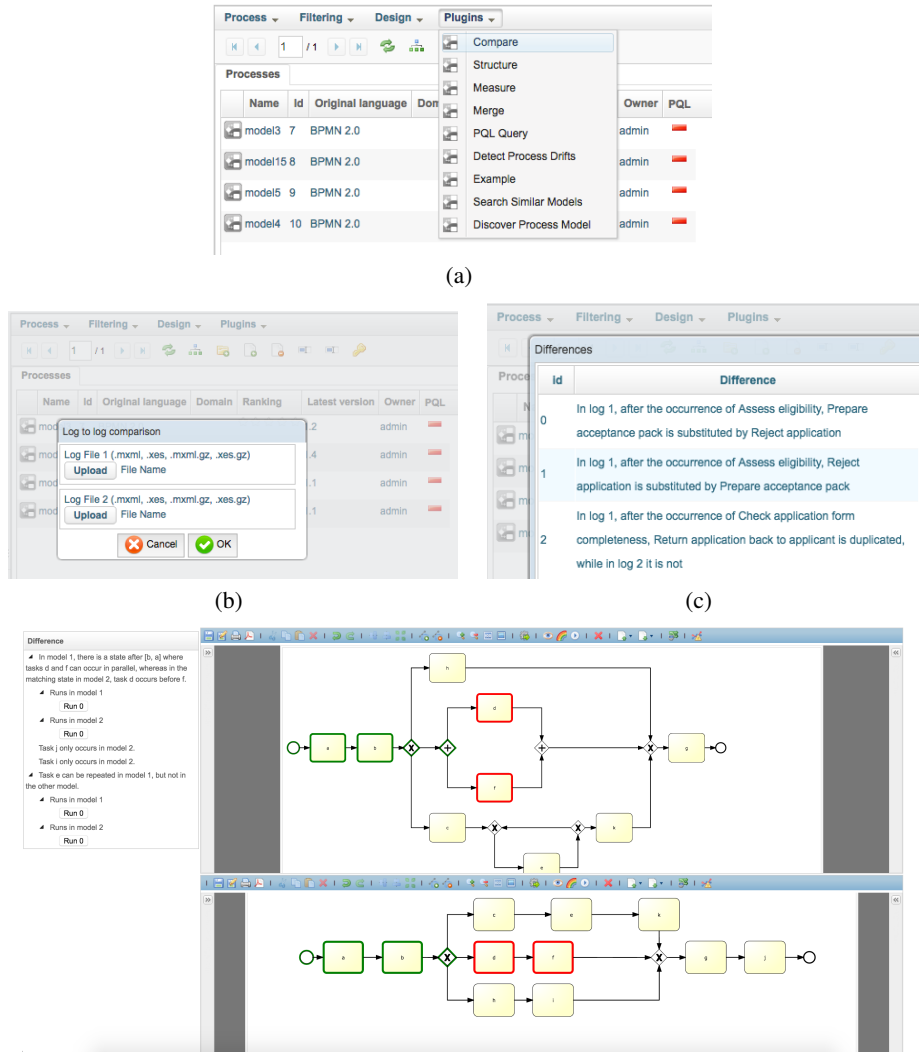


Fig. 2: Apromore Compare interface.

Given a pair of process artifacts, the steps for the three types of comparison are: 1. computation of event structures out of event logs and/or models, 2. comparison of event structures, and 3. verbalization of the identified differences. In the current implementation the type of event structures used is *prime event structures* [6], the comparison is performed using a so called *partial synchronized product* [4], and the verbalization of the differences is aligned with the corresponding papers [1, 3–5].

Figure 2 shows the different user interfaces in Apromore for the comparison of two logs and two models. Figure 2(a) shows the menu in Apromore where the *Compare*

operation is located, Figure 2(b) depicts the input dialog to upload a pair of event logs and Figure 2(c) shows the popup window displaying the statements in natural language explaining the identified differences. Finally, Figure 2(d) depicts the representation of the differences resulting from the comparison of two models. In this case, the differences are represented in two ways: 1. as statements in natural language (left) and 2. as overlaid graphics on the process models (right).

2 Significance and Maturity

Our toolchain exhibits a novel approach for identifying root causes of deviant process executions (via event log comparison), identifying differences between process executions and normative process specifications (conformance checking of an event log with a process model), and identifying differences between different versions or specifications of process models (i.e. comparing two different business process models).

The approach can be applied both in intra-organizational and cross-organizational settings. For instance, different process variants and executions in public organizations can be identified and analyzed to obtain a set of generic models (e.g. including best practices) and a set of additional organization-specific features. In addition, process executions in different organizational branches can be analyzed to identify causes for performance differences.

This approach provides for the first time a tool that abstracts from the representation (i.e. process model or event log) and is capable of reporting a complete set of differences via natural language statements. The set of produced statements are compact and interpretable to allow for a clear overview of differences between process models and/or logs of process executions. The reporting of differences is specifically intended to inform business analysts, and comparison of complex models and logs is, as such, no longer limited to technical users only. Consequently, users who are directly involved in the business process under investigation are now able to interpret the results of the comparison. Furthermore, the results provided are complete (i.e. all behavioral differences are identified), more compact and precise than existing approaches and provide, therefore, a much better assessment of differences between process models and logs.

The Apromore features used in this approach have been evaluated extensively with respect to accuracy, scalability and advantages over existing approaches. The evaluations comprised large collections of both artificial and real-life process models and event logs. The qualitative evaluation showed that the presented toolchain produces a more compact and much more understandable diagnosis than existing techniques. Furthermore, the tool exposes differences that are difficult or impossible to identify otherwise. The quantitative evaluation involved over 700 real-life process models and showed that the proposed approach has reasonable execution times (within seconds). Even in extreme cases with a high number of differences between the process model and the event log (with the event log containing more than 8,000 event occurrences, considering distinct traces only), the execution time is still within a few minutes. The detailed results of these evaluations are reported in [3] (log2log), [1] (model2log) and [4, 5] (model2model).

Apromore features an OSGi plugin framework to support dynamic enabling/disabling of plugin bundles and multiple bundle versions. These capabilities include presentation capabilities with respect to process model restructuring, filtering of models based on process-related aspects, searching and querying for specific process patterns, advanced design of process models, including configuring and merging if existing models, and evaluation capabilities to assess the quality and correctness of models, as well as simulation and conformance checking techniques for benchmarking. Furthermore, Apromore provides full import and export functionality to a large variety of business process modeling languages and log formats, such as BPMN, XPDL, EPML, ARIS, YAWL, PNML, XES and MXML.

Apromore is the result of over six years of ongoing development and is currently in version 3.4. The platform is implemented via a service-oriented architecture and deployed as a Software as a Service. The technologies used in Apromore combine Spring as the Java development framework, Maven as the dependency manager, OSGi as the plugin architecture, EclipseVirgo as the OSGi-based application server, and ZK as the AJAX front end. The chosen technologies allow Apromore to be an extensible framework, where new plugins can be easily added to an ecosystem of advanced capabilities for managing process model collections.

3 Screencast

A screencast of Apromore's compare feature can be found at `http://goo.gl/JB1EDv`. The public release of Apromore is available at `http://apromore.qut.edu.au` and its source code can be downloaded under the GNU LGPL license version 3.0 from `https://github.com/apromore/ApromoreCode`.

References

1. García-Bañuelos, L., van Beest, N.R.T.P., Dumas, M., La Rosa, M.: Complete and interpretable conformance checking of business processes. QUT ePrints, Technical report #91552, http://eprints.qut.edu.au/91552/ (December 2015)
2. García-Bañuelos, L., van Beest, N.R.T.P., Dumas, M., La Rosa, M.: Business process conformance checking based on event structures. In: Proc. of NWPT'2015, Reykjavik University (2015) 3 pages.
3. van Beest, N.R.T.P., Dumas, M., García-Bañuelos, L., La Rosa, M.: Log delta analysis: Interpretable differencing of business process event logs. In: Proc. of BPM 2015, Springer (2015) 386–405
4. Armas-Cervantes, A., Baldan, P., Dumas, M., García-Bañuelos, L.: Diagnosing behavioral differences between business process models: An approach based on event structures. Information Systems **56** (2016) 304–325
5. Armas, A., Baldan, P., Dumas, M., García-Bañuelos, L.: Behavioral comparison of process models based on canonically reduced event structures. In: BPM 2014, Springer (2014) 267–282
6. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri Nets, Event Structures and Domains, Part I. TCS **13** (1981) 85–108
7. Dumas, M., García-Bañuelos, L. In: Process Mining Reloaded: Event Structures as a Unified Representation of Process Models and Event Logs. Springer (2015) 33–48

BPMN Miner 2.0: Discovering Hierarchical and Block-Structured BPMN Process Models

Raffaele Conforti[1], Adriano Augusto[1], Marcello La Rosa[1], Marlon Dumas[2], and Luciano García-Bañuelos[2]

[1] Queensland University of Technology, Australia
{a.augusto, raffaele.conforti, m.larosa}@qut.edu.au
[2] University of Tartu, Estonia
{marlon.dumas, luciano.garcia}@ut.ee

Abstract. We present *BPMN Miner 2.0*: a tool that extracts hierarchical and block-structured BPMN process models from event logs. Given an event log in XES format, the tool partitions it into sub-logs (one per subprocess) and discovers a BPMN process model from each sub-log using existing techniques for discovering BPMN process models via heuristics nets or Petri nets. A drawback of these techniques is that they often produce spaghetti-like models and in some cases unsound models. Accordingly, BPMN Miner 2.0 applies post-processing steps to remove unsound constructions as well as a technique to block-structrure the resulting process models in a behavior-preserving manner. The tool is available as a standalone Java tool as well as a ProM and an Apromore plugin. The target audience of this demonstration includes process mining researchers as well as practitioners interested in exploring the potential of process mining using BPMN.

Keywords: Process Discovery, BPMN, Structured Process Models, Hierarchical Process Models, ProM, Apromore

Process mining is a discipline within Business Process Management that aims to extract actionable insights from process execution logs [9]. Such execution logs, called *event logs* for short, can be extracted from common or special-purpose IT systems available in today's organizations, such as an ERP system or a Claims Management System.

One particular category of process mining methods is that of *automated process model discovery*. These methods extract a process model from an event log. Real-life logs, however, are typically affected by noise (e.g. in the form of infrequent behavior) and are incomplete (i.e. they do not contain all possible behavior of the business process under analysis). Consequently, process model discovery is not a trivial procedure and leads to process models of varying quality, which often approximate the actual business process, depending on the discovery algorithm that is employed.

A large number of automated process discovery algorithms have been proposed. These algorithms strike various tradeoffs between accuracy (measured in terms of fitness, precision and generalization [9]) and *complexity* (measured in terms of model size and other complexity metrics [7]). Some algorithms tend to discover process models with high accuracy at the cost of high model complexity. The importance of model

complexity should not be underestimated as this underpins the understandability of the extracted process model, and so ultimately its value to users. In this respect, empirical studies [5] have shown that besides model size, an important proxy for process model understandability is the *structuredness* of the model. This latter observation has led to the design of discovery algorithms such as the *Inductive Miner* [6] and the *Evolutionary Tree Miner* [2], which discover structured process models by design. Models discovered using such algorithms may be further away from reality depending on the degree of unstructuredness of the actual business process the log refers to. For example, the Inductive Miner tends to over-generalize the behavior in the log, leading to low precision while maximizing fitness [1]. On the other hand, discovery algorithms such as the *Heuristics Miner* [11] and the *Fodina Miner* [10] tend to strike better results in terms of accuracy but produce more complex and sometimes syntactically incorrect and unsound process models [1]. Furthermore, the majority of discovery algorithms produce models that are represented in languages that are either not widely accepted among practitioners (e.g. Petri nets), too technical (e.g. Heuristics nets) or too abstract and over-generalizing (e.g. fuzzy nets or process maps).

The *BPMN Miner* algorithm [3, 4] is designed to address the above limitations. This algorithm discovers models in the BPMN 2.0 language [8], a *de jure* standard widely supported by vendors and practitioners. Moreover, by exploiting implicit functional and foreign-key dependencies between attributes in the event log, the algorithm can generate hierarchical BPMN models, i.e. models organized over two or more abstraction levels via the use of subprocess models. In order to further reduce the complexity of the discovered models, the algorithm exploits an extensive set of notational elements provided by the BPMN language, such as boundary events (to model interrupting exceptions), activity markers (loops and multi-instance) and event subprocesses.

BPMN Miner relies on existing (*baseline*) automated process discovery algorithms to generate an initial model of each subprocess. The baseline algorithms supported are Inductive Miner, Heuristics Miner, Fodina, ILP Miner and Alpha Miner. Version 2.0 of BPMN Miner additionally embeds an algorithm that discovers sound and maximally-structured BPMN models, namely the *Structured Miner* [1]. This latter algorithm removes unsound and unstructured constructs in the discovered model, thus further increasing its potential usability.

The minimum input required by the tool is the log from which to discover a BPMN model. By default, the Heuristics Miner is chosen as the baseline discovery algorithm. It is however possible to customize a number of input parameters in order to tune the results (see Figure 1).

Besides the baseline discovery algorithm, one can choose whether the partitioning of the log into sublogs is achieved via a noise tolerant dependency discovery algorithm, or not, in which case the log is assumed to be noise-free. Additionally, it is possible to sort the input log based on the timestamp of its events, and to switch on the structuring of the discovered process model. The latter function is only effective when the baseline discovery algorithm does not already produce a structured model by design.

Additional parameters can be set to fine-tune the discovery of BPMN-specific notational elements, on the basis of a number of heuristics. For example, one can set tolerance levels for the identification of boundary timer and message events, and of multi-instance activity markers.

Once these parameters are set, the algorithm retrieves event attributes which may be used as primary keys for the partitioning of the log into sublogs. In this context, a primary key is an attribute which is recurrent in all events that are related to the activities of a particular subprocess. For example, an attribute "invoice" would be recurrent across all events related to the handling of the invoice, as part of an overarching order-to-cash process. All such events that are related to the handling of the invoice would be isolated in a separate sublog, from which the corresponding subprocess for handling invoices will then be discovered. The user has the possibility of steering the use of particular event attributes by selecting/deselecting them from a list that is automatically populated by the tool.

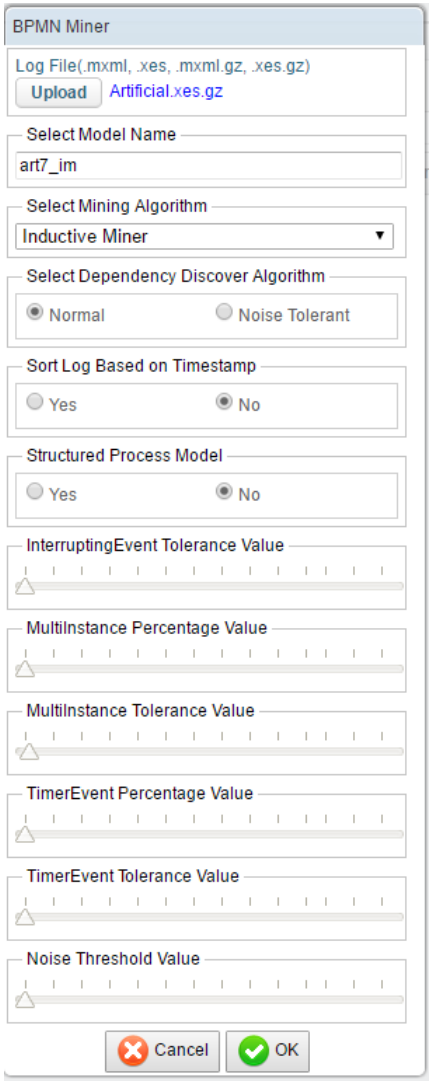


Fig. 1. Parameters.

Next, the algorithm assigns a specific primary key to each sublog. If more than one primary key can be assigned to the same sublog, the user is asked to choose from a droplist, where the first key is the most fitting one. At this point the event log is partitioned into sublogs using the chosen primary keys, and each sublog is passed as input to the baseline discovery algorithm for model discovery. When the discovery has completed, each subprocess model is structured separately, if this option is enabled, and assembled together as part of a single hierarchical BPMN model.

Figure 2 shows an example of hierarchical process model discovered by BPMN Miner, which was fully structured after the discovery. This model exhibits two expanded subprocesses (one with loop marker, the other with multi-instance marker) and one event subprocess. Figure 3 shows an example of hierarchical process model discovered with BPMN Miner in ProM. This latter model is maximally structured and exhibits two expanded subprocesses (again, with loop and multi-instance markers), two event subprocesses (one nested within a subprocess), and boundary timer and message events with exception flows. Both models were discovered from synthetic log of an order-to-cash process.

The accuracy and scalability of BPMN Miner have been extensively evaluated using over 600 event logs, including both artificial and real-life event logs. The majority of the artificial logs were generated from the SAP R/3 and IBM BIT process model collections, which group models from a variety of domains, including finance, sales,

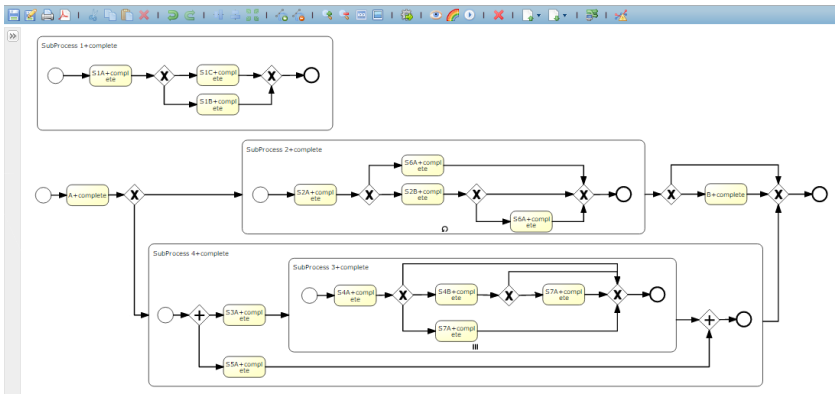


Fig. 2. Example of fully-structured process model discovered by BPMN Miner in Apromore.

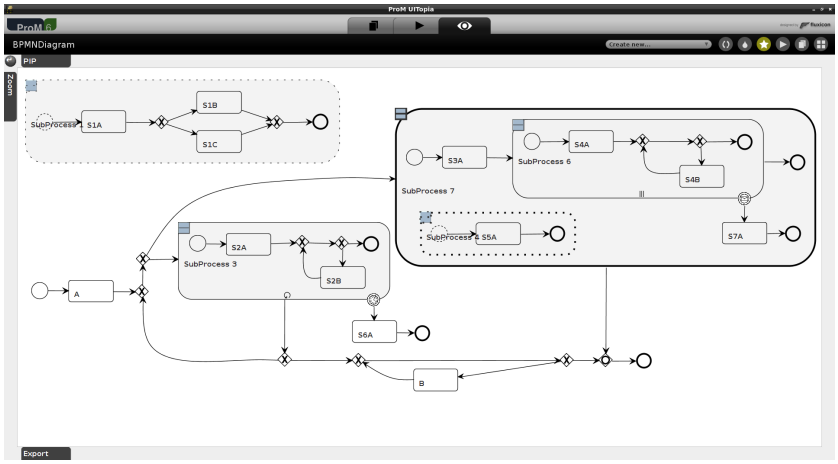


Fig. 3. Example of maximally-structured process model discovered by BPMN Miner in ProM.

accounting, logistics, communication and human resources. The results of these evaluations are reported in [3, 4, 1].

The results of the experiments show that the tool scales well to large and noisy real-life logs, performing within reasonable time bounds in the order of minutes. Moreover, the results indicate a statistically significant improvement of discovery accuracy and model complexity over all the baseline discovery algorithms supported by the tool.

BPMN Miner 2.0 is available as an OSGi plugin of the Apromore process model repository, as a plugin of the ProM Framework, as well as a standalone commandlineJava tool. Apromore is an online open-source ecosystem of advanced capabilities for managing large process model collections, including process modeling, simulation, filtering, querying, similarity search, behavioral comparison and model merging. ProM is the largest on-source process mining framework, offering over 300 plugins (in its latest incarnation) offering process model discovery, conformance checking, variants and deviance mining, and log analysis capabilities.

A screencast is available at https://youtu.be/eb0k2RO2PQ8. This video illustrates different examples and provides a brief explanation of the tool settings along with the possible outputs that can be obtained by varying the input parameters. BPMN Miner 2.0 is embedded as an OSGi plugin in the online platform Apromore, which has been used for the screencast (http://apromore.qut.edu.au). The artificial log used in the screencast is available at https://goo.gl/AdvnEd.

BPMN Miner is also available as a ProM plugin (http://promtools.org) and as a standalone Java tool (http://apromore.org/platform/tools).

References

1. A. Augusto, R. Conforti, M. Dumas, M. La Rosa, and G. Bruno. Automated discovery of structured process models: Discover structured vs. discover and structure. In *Proc. of ER*. Springer, 2016. Preprint available at http://eprints.qut.edu.au/95189/.
2. J. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. On the role of fitness, precision, generalization and simplicity in process discovery. In *Proc. of CoopIS*, volume 7565 of *LNCS*, pages 305–322. Springer, 2012.
3. R. Conforti, M. Dumas, L. García-Bañuelos, and M. La Rosa. Beyond tasks and gateways: Discovering BPMN models with subprocesses, boundary events and activity markers. In *Proc. of BPM*, LNCS, 2014.
4. Raffaele Conforti, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. Bpmn miner: Automated discovery of bpmn process models with hierarchical structure. *Information Systems*, 56:284–303, 2016.
5. M. Dumas, M. La Rosa, J. Mendling, R. Mäesalu, H.A. Reijers, and N. Semenenko. Understanding business process models: the costs and benefits of structuredness. In *Proc. of CAiSE*, volume 7328 of *LNCS*, pages 31–46. Springer, 2012.
6. S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Discovering block-structured process models from event logs - a constructive approach. In *Proc. of PETRI NETS*, volume 7927 of *LNCS*. Springer, 2013.
7. J. Mendling. *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*. Springer, 2008.
8. Object Management Group (OMG). *Business Process Model and Notation (BPMN) ver. 2.0*. Object Management Group (OMG), January 2011.
9. W.M.P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.
10. S.K.L.M. vanden Broucke, J. De Weerdt, J. Vanthienen, and B. Baesens. Fodina: a robust and flexible heuristic process discovery technique. http://www.processmining.be/fodina/. Last accessed: 03/27/2014.
11. A.J.M.M. Weijters and J.T.S. Ribeiro. Flexible Heuristics Miner (FHM). In *Proc. of CIDM*. IEEE, 2011.

Interactively Exploring Logs and Mining Models with Clustering, Filtering, and Relabeling

Xixi Lu, Dirk Fahland, Wil M.P. van der Aalst

Eindhoven University of Technology, The Netherlands
{x.lu,d.fahland,w.m.p.v.d.aalst}@tue.nl

Abstract. Real-life event logs often contain many data quality issues, which obstruct existing discovery algorithms from discovering meaningful process models and process analysts from conducting further process analysis. In this paper, we present an integrated tool that provides support for dealing with three of these data issues: logs comprising recordings of multiple heterogeneous variants of a process; traces containing multitude of deviating events in an infrequent context; event labels being imprecise. The tool is called *Log to Model Explorer* and helps users in interactively and iteratively exploring and preprocessing a log by clustering, filtering and event relabeling, enabling them to discover more meaningful process models.

Keywords: Process Mining, Log Exploration, Log Preprocessing, Trace Clustering, Log Filtering, Duplicated Tasks, Event Label Refinement, Process Discovery

1 Exploring Log for Complementing Process Discovery

Process Mining takes as input an event log that contains event data about past process executions. Real-life event logs may have many data quality issues. For example, they often contain various heterogeneous variants of the process. Every case going through the process may have some deviating events. Moreover, events may have imprecise and ambiguous labels. These data quality issues cause process discovery algorithms to fail in extracting meaningful process models. Although many methods and techniques have been proposed for handling some particular data quality issues, no integrated, explorative approach and support for preprocessing an event log has been proposed yet.

We discuss the use case using the the Road Traffic Fine log [1]. A user may start with applying a discovery algorithm on the log. However, due to aforementioned data quality issues, the user could obtained a rather complex model unsuitable for understanding the process. Fig. 1 shows a discovered model which contains a large loop in which each activity can be either executed or skipped, suggesting that in this part activities have been executed arbitrarily. However, there is no (set of) traces in the log showing this arbitrary behavior. Rather the log contains multiple different variants that are shown in Fig. 4. In order to identify these variants, the user would now have to pre-process the

Fig. 1: Model discovered with Inductive Miner from a log with data quality issues.

log (by choosing from various techniques), then discover a model and evaluate whether the model is good or further/different preprocessing is required. User would have to try dozens clustering plugins, filter the log in various ways and apply multitude discovery algorithms, not to mention the parameters he/she need to set when trying them. This makes exploring a log a rather laborious task, especially if the user does not know where the issue lies and the methods he needs may be distributed over different tools. Support for interactively exploring a log for discovering suitable models is missing.

This paper describes the plugin *Log to Model Explorer* in the *TraceMatching* package of the Process Mining framework ProM[2]. The plugin allows users explore different views on a log by integrating the support for three main preprocssing functionalities: clustering traces into variants, filtering infrequent behavior, and refining imprecise labels. The tool immediately shows the result of each preprocessing step by for example visualizing the model discovered. This allows user interactively and iteratively explore the log and discover models of interest. It integrates our previous work on clustering, filtering [2] and relabeling techniques [3] for control-flow behavior.

The aim of this demo paper is to demonstrate the basic concepts of the *Log to Model Explorer* and illustrate its general functionality and suitable use cases. We demonstrate our ideas on a random sample of 1000 traces of the Road Traffic Fine (RTF) log [1]. A screencast which demonstrates the features of the plugin and how it can be used in practice using the same log can be downloaded[3]. In Sect. 2, we first give an overview of the tool and explain the three main functionalities: clustering, filtering and relabeling. In the end, we discuss related tools and future work as we conclude.

2 The *Log to Model Explorer*

Our tool integrates three pre-processing operations on the log (clustering, filtering and relabeling) with immediate discovery of a model on (parts of) the pre-processed log. Fig. 2 shows an overview of possible paths comprising these four steps. Given an input log, the tool starts with clustering the traces based on their behavioral similarity (1). By selecting a cluster, the user is shown the corresponding sublog or the model discovered on this sublog (4). Optionally, the user may apply a filter on this sublog to remove infrequent events (2) and view the result of such filter (4). Next, for a cluster, the user may choose to refine event labels (3) while exploring different representations of models (4). After refining the labels, the user may decide to cluster the relabeled sublog again, and so on, iteratively. Fig. 3 shows the main GUI of the tool, highlighted with the enumerated sections that correspond to the aforementioned steps. Both logs and models at any stage can be exported for further analysis.

[2] Availabel in the ProM nightly builds under: `http://promtools.org`

[3] `https://svn.win.tue.nl/repos/prom/Documentation/TraceMatching/2016demo.mp4` or watch online at `https://vimeo.com/176721009`

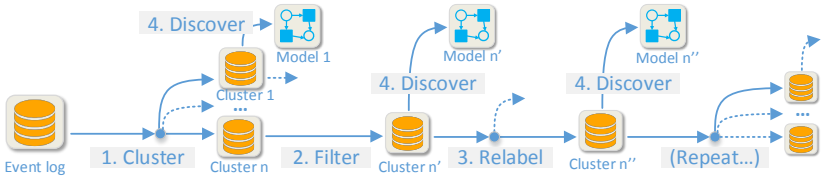


Fig. 2: Possible interactive exploration paths in the *Log to Model Explorer*.

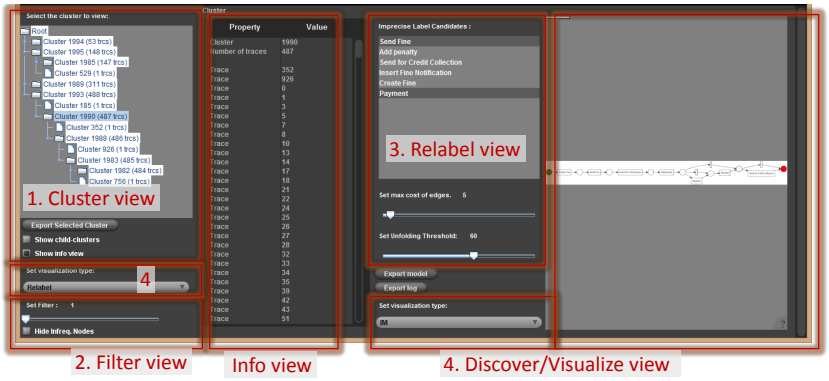


Fig. 3: The main GUI of the *Log to Model Explorer* plugin in the ProM Framework.

Trace Clustering. Trace clustering is an essential step of log-preprocessing to identify variants of the process in the log that are behaviorally similar [4]. A sublog of behaviorally similar traces is more likely to yield a more coherent and structured model. We use the clustering technique of [2] that measures behavior similarities of traces and builds a hierarchy of clusters, which we visualize as a tree in the cluster view (1) as shown in Fig. 3. The root node has a set of main clusters as its children. Each main cluster can be further expanded to view its sub-clusters. The user may select any (sub-)cluster and explore the cluster further. When a user selects a cluster, the plugin invokes the current visualizer, for example the Inductive Miner [5], which immediately shows the model descovered that represents this part of the log. It is also possible to view the selected cluster and its child-clusters together by checking the radio-button "*Show child-clusters*". For example, Fig. 4 shows that the user selected cluster 1982; the right-hand side of the screen shows at the top the model discovered for cluster 1982 and below the models for the two children of cluster 1982, i.e., the sub-variants. When inspecting the two sub-clusters, we see that the first one shows multiple executions "*Payment*" (in a loop) whereas the second sub-cluster shows no "*Payment*" has been executed; the model of the parent cluster 1982 combines both into a loop that can be skipped.

Context-Aware Event Filtering. Next, the user may decide to filter *infrequent behavior*, sometimes interchangeably called *deviations* or *noise*, from the traces of a cluster to reveal the main behavior of the cluster. This filtering technique is based on the deviation/noise detection described in [2] which does not requiring any normative process model. We consider an event e of activity A *infrequent* (uncommon, deviating) if e has a context of preceding and succeeding events that is different from most other occurrences

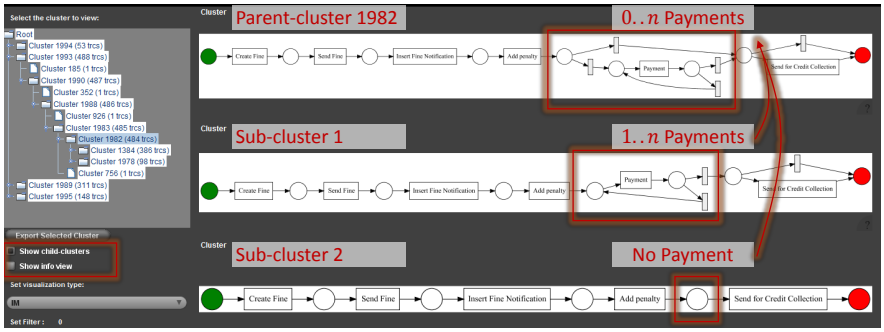


Fig. 4: Interactive, visual comparison of a cluster and its sub-variants.

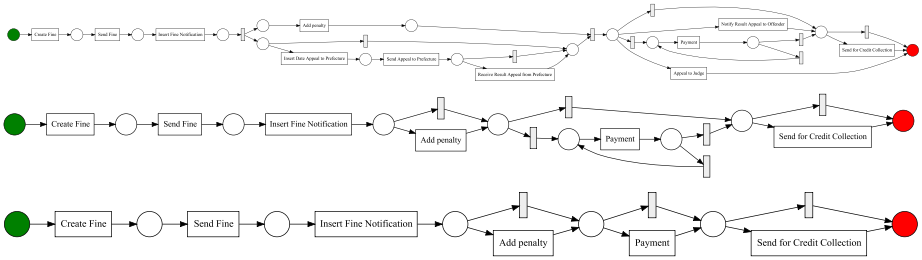


Fig. 5: The models discovered by filtering events based on the frequency of their context for 0%, 1% and 5% threshold.

of activity A. For example, assume a log with 99 traces $\langle abcd \rangle$ and 1 trace $\langle abdbde \rangle$; activity b has two contexts, between a and c, and between c and d; the latter one is infrequent. In a similar way, the single occurrence of e can be classified as infrequent. Our plugin provides a filter to remove all such infrequently classified events below a user-chosen threshold. The filtered log only shows the frequent behavior. Selecting a cluster (1993) in the RTF sample, we respectively obtain three models shown in Fig. 5 by filtering the cluster by 0%, 1% and 5%. Note that when setting the threshold to 5%, the loop around "*Payment*" is removed, revealing that less than 5% of the cases have a second "*Payment*".

Event Relabeling. In addition to clustering and filtering, we also provide support for refining event labels. Relabeling events allows user explore different representations of the same log, in particular when events of the same activity have clearly different contexts that are equally frequent. For example, assume the traces $\sigma_1 = \langle abcd \rangle$ and $\sigma_2 = \langle abcbde \rangle$ are equally frequent. Rather than filtering out the second b in σ_2, the user may want to view and analyze all behavior in the log. Giving the log as-is to a discovery algorithm would introduce a loop for repeated execution of b, but also several skip steps to allow d to occur after c (as in σ_1) and after b (as in σ_2), thus introducing many more behaviors. Alternatively, our plugin detects that the second b in σ_2 has a different context and allows to relabel it to b_2 allowing to discover a more precise model [3]. The user can influence the amount of relabeling by threshold parameters. Together with a discovery algorithm that guarantees discovering a fitting model, the user

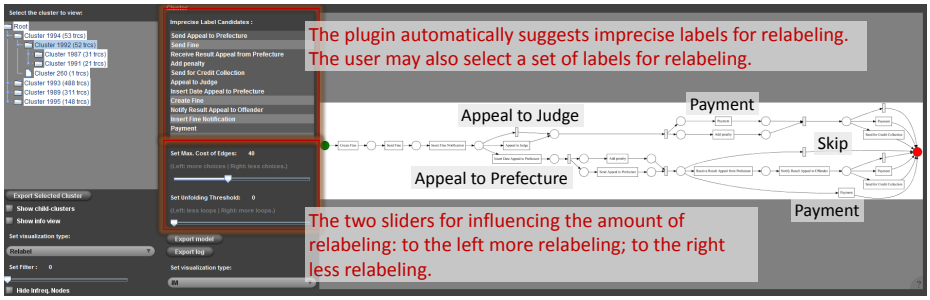


Fig. 6: Identifying two different variants within a process through event relabeling.

may now explore different models for the (sub)log that are all fitting but have different precision (generalization). Taking the RTF sample, the relabeling allowed us to find the duplicated tasks ("*Payment*", "*Add pentaly*" and "*Send for Credit Collection*") and to discover two alternative paths in the model as shown in Fig. 6: if offenders decide to "*Appeal to Judge*", then there is always a "*Payment*"; if offenders decide to "*Appeal to Prefecture*", then there are alternatives to skip "*Payment*". User may export discovered models, relabeled logs or original logs of cluster in ProM for further analysis.

Conclusion, Limitation and Future Work. In this paper, we presented the *Log to Model Explorer* as an integrated tool for log preprocessing and discovering more suitable models for a log. We showed that the tool supports three main functionalities: trace clustering, infrequent event filtering, and event label refinement. Currently, commercial tools such as Disco and Celonis[4] have extensive support for filtering a log for obtaining a variant but require the user to know which variant he/she want to have. Furthermore, these tools also suffer from the imprecise label problems and often discover spaghetti-like process maps. Available academic tools focus on solving one particular data quality issue; using them iteratively is tedious. A limitation of our tool is its performance in handling large logs; the running time scales polynomial in number of events. Currently, random sampling is used for improving the performance. As next step, we plan to generalize the tool into a framework for exploring an event log, allowing different clustering, filtering and log visualization techniques to be integrated.

References

1. de Leoni, M., Mannhardt, F.: Road Traffic Fine Management Process. Technical report, Eindhoven University of Technology (2015)
2. Lu, X., Fahland, D., van den Biggelaar, F.J.H.M., van der Aalst, W.M.P.: Detecting deviating behaviors without models. In: BPM 2015, Workshops, Springer (2015) (to appear)
3. Lu, X., Fahland, D., van den Biggelaar, F.J.H.M., van der Aalst, W.M.P.: Handling duplicated tasks in process discovery by refining event labels. In: BPM 2016, Springer (2016) (to appear)
4. Greco, G., Guzzo, A., Pontieri, L., Saccà, D.: Discovering expressive process models by clustering log traces. IEEE Trans. Knowl. Data Eng. **18**(8) (2006) 1010–1027

[4] Disco :`https://fluxicon.com/disco/`; and Celonis : `http://www.celonis.de/`

5. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. In: Application and Theory of Petri Nets and Concurrency. (2013) 311–329

SHAPEworks: A BPMS Extension for Complex Process Management

Saimir Bala[1], Giray Havur[1], Simon Sperl[2], Simon Steyskal[1,2], Alois Haselböck[2], Jan Mendling[1], and Axel Polleres[1]

[1]Vienna University of Economics and Business, Austria
{name.surname}@wu.ac.at
[2]Siemens AG, Austria {name.surname}@siemens.com

Abstract. Complex engineering projects, such as the deployment of a railway infrastructure or the installation of an interlocking system, involve human safety and make use of heterogeneous data sources, as well as customized engineering tools. These processes are currently carried out in an ad-hoc fashion, relying on the experience of experts who need to plan, control, and monitor the execution of processes for delivering value to the customers. This setting makes an automated overarching-process a crucial step towards supporting engineers and project managers to deal with safety-critical constraints and the plethora of details entailed by the process. This paper demonstrates a tool that combines methods from automatic reasoning, ontologies and process mining, implemented on top of a real Business Process Management System (BPMS).

Keywords: Process Management, Resource Management, Ontologies, Process Mining, BPMS, Compliance

1 Introduction and Significance to BPM

Complex engineering projects involve a wide variety of tools, data-sources and specific domain knowledge. This is the case, e.g., for the installation of a railway interlocking system. Furthermore, these projects must respond to strict constraints imposed by safety rules and regulations in the operating domain. Managing and monitoring these projects in real life presents a number of challenges, especially in companies with a high focus on optimizing their operations through Business Process Management (BPM).

Three perspectives are of crucial importance in our setting. First, resources need to be allocated optimally not only to optimize delivery times of the final product, but also to comply with the safety rules in the domain, e.g., assign resources to tasks according to their expertise level. Second, the engineers' work must be traced in order to allow internal or external auditors to verify it at a later stage. This work can be reflected by generated artifacts, such as emails, word processor documents, and Version Control System (VCS) logs, which result from the engineering tools used by the engineers to accomplish their tasks. Third, the data from different software tools and engineering

workflows must be accessible in order to ease reporting and documentation of what work has been done. To respond to these challenges, an overarching engineering process is fundamental, along with a framework that supports its execution [3].

We have devised a framework for addressing the above mentioned perspectives through a single automated solution. We demonstrate how automated reasoning can be used for optimally scheduling resources and tasks. The resulting schedule is flexible and can be customized by imposing soft constraints (e.g., optimization statements) or hard constraints (e.g., compliance rules) on resources. We use ontologies to represent organizational and other process data, e.g.,, rules and regulations. History logs are frequently monitored and mined in order to update existing knowledge of the process with new insights. Furthermore, we support document generation, which helps towards reducing human errors and increasing compliance to documentation requirements.

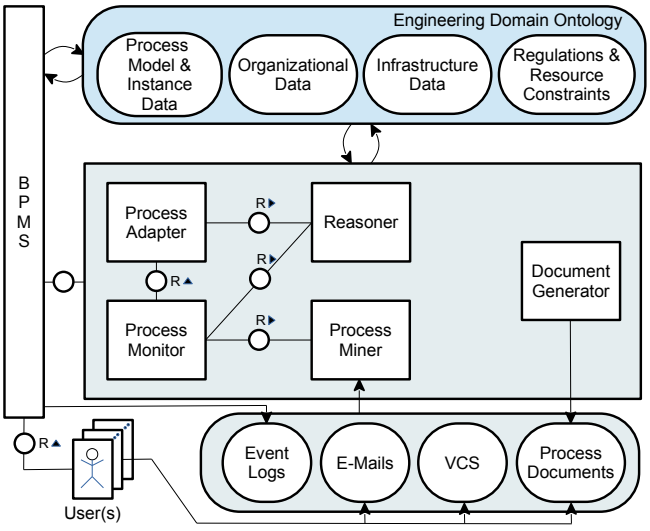


Fig. 1: Framework for process management in complex engineering projects

Fig. 1 illustrates the developed framework, using the Fundamental Modeling Concepts (FMC) notation.[2] The main components are briefly described below.

BMPS. We have extended the Camunda BPM engine, an extensible business process engine which offers API access. The engine executes the defined engineering process, and we interact with it by capturing the events about task completion, process start, process end, etc, and by using historical information that is stored in its logs.

Reasoner. The reasoner module is in charge of both computing resource allocations using Answer Set Programming (ASP) and validating Shapes Constraint Language (SHACL) [5] constraints for potential violations of domain constraints. Details on the resource allocation approach can be found in [4].

[2] http://www.fmc-modeling.org/

Process Monitor. This component listens to task completion or task starting events, in order to actively check for process non-compliant behavior and to signal potential anomalies. The process monitor uses results from the Miner to raise alerts in case the process could not be executed according to the schedule.

Miner. The Miner deals with historical data in order to infer useful information about the process. In our use case, the focus is on gathering useful statistics about resources and process activities.

Document Generator. The Document Generator is able to create or fill in textual documents from data that has been generated by users that complete their tasks. The data includes user comments to tasks and process variables.

Process Adapter This component is in charge of computing adaptations when slight deviations occur. When the adaptation is not possible, it triggers an alert to the Reasoner and the process must be stopped and re-planned.

We use an ontology to represent the engineering domain and the organizational (i.e., resource-related) knowledge, business processes, and regulations and policies. Process relevant data are stored in RDF, which allows also to use SHACL to check compliance constraints. The ontology can be used as a shared repository by the above mentioned components to share data. Our framework also considers data from event logs, emails and VCS, which can be further used for mining traditional processes, text, and project-oriented processes [1].

2 Use Case and Tool Description

We have extended the Camunda[3] BPMS with the components described in Sect. 1. Our solution stems from a real industry scenario. The bigger context of the tool is the setting described in [2]. Here we show how our tool can be used to support a typical engineering process.

2.1 Industry Scenario

A typical process in the railway domain aims at the release of a new engineering system for a railway customer. The process starts when a new agreement with a client has been signed by the project management team. A new repository is created for the customer data and, at the same time, possible additional data are requested from the client. The next step is the actual *System engineering* activity, where the new system is built. In turn, the lab management team sets up the laboratory for executing possible required tests. If test results are not satisfactory, the system must be re-engineered. Otherwise, a report is generated and the work is handed over again to the project management team, which then delivers a new release to the customer. Fig. 2 illustrates such a process, modeled in Business Process Modeling Notation (BPMN).

[3] https://camunda.org/

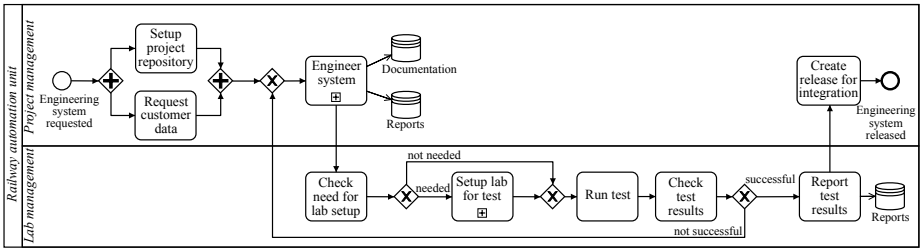


Fig. 2: An engineering process in the railway domain

2.2 Usage of the Tool

In this section we show a use case of our tool in the above mentioned scenario. Initially the user logs in to the Camunda BPMS where they can see the task list. The users with administrator privileges can start a new engineering process. When the process is started and before the first activity is executed, the reasoner computes a schedule. Afterwards, the user interacts with the results of the reasoner by confirming or modifying the schedule. A Graphical User Interface (GUI), shown in Fig. 3, has been implemented for this purpose. It shows the assignments of resources to activities. This is assisted by lock icons on the GUI, providing constraints on the resources who must be assigned to a particular task. For example, the project manager may enforce the *Check test results* activity to a particular resource.

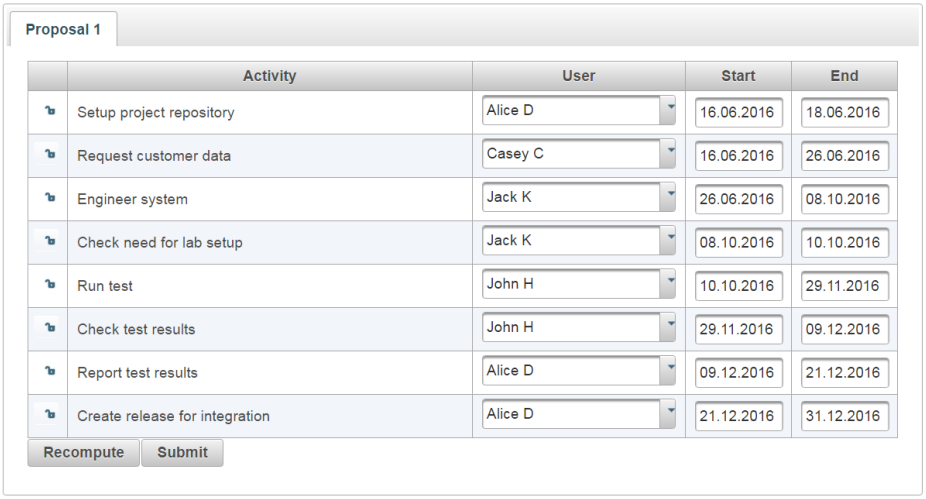


Fig. 3: Automatic resource assignment, as a result of the scheduling component

After the schedule is confirmed, the allocations take place and the resources can see the tasks appearing in their tasklist. Then, the process can be executed while the process-monitoring component continuously listens to events that occur during the execution of the process. In particular, when a task is finished, the process monitor checks whether

the schedule is respected. At the same time, the mining-component updates statistical information from the Camunda logs. For example, after a task is completed, a skill score and an expertise score are updated for the allocated resource. Moreover, statistics are collected, e.g., punctuality of task completion, standard deviation, average duration, percentage of deviations, etc.

3 Maturity and Future Work

SHAPEworks is our first implementation in the context of a BPM scenario which demands for more complexity. Currently it is a prototype that serves mainly as a proof-of-concept. It shows the advantage of having an integrated solution of different approaches implemented on top of a BPMS. We plan to further develop our extension by adding more features, divided into three levels.

Level 1 (current). SHAPEworks includes: *i)* resource (re-)allocation with data and resources from Camunda; *ii)* ontology describing the organizational model; *iii)* process monitoring that triggers alerts when the process risks running late and the process execution does not respect the allocation; and *iv)* mining history of tasks and resources, and updating ontology with mined data.

Level 2. SHAPEworks includes: *v)* allocation of non-human resources, which are fully synchronized with the data from the ontology; *vi)* delay prediction by mining history logs; and *vii)* infrastructure model stored in the ontology.

Level 3. SHAPEworks includes: *viii)* flexible resource (re-)allocation, i.e., allocating resources up to the next decision point of the process, thus enabling a more dynamic schedule; *ix)* mining XOR probabilities, i.e., the likelihood of particular choices made in XOR gates; and *x)* feasibility check using XOR probabilities, i.e., given the likelihood of the path to be executed in the business process and the resources, check if the execution is possible in n amount of time.

The Camunda BPMS along with our custom extensions has been deployed on a server and can be used by following this link: http://camunda.ai.wu.ac.at:8080/camunda. Credentials for project managers – administrative users with higher privileges – are ‘demo’ and ‘demo’, respectively user name and password; standard users can access with username same as their name and the string ’password’ as password. A screencast of the tool can be found in http://camunda.ai.wu.ac.at/shapeworks/video.html.

Acknowledgements. This work is funded by the Austrian Research Promotion Agency (FFG) under grant 845638 (SHAPE): http://ai.wu.ac.at/shape-project/

References

1. Bala, S., Cabanillas, C., Mendling, J., Rogge-Solti, A., Polleres, A.: Mining Project-Oriented Business Processes. In: BPM. pp. 425–440 (2015)
2. Cabanillas, C., Mendling, J., Polleres, A., Haselböck, A.: Safety-critical human-and data-centric process management in engineering projects. In: 5th International Symposium on Data-Driven Process Discovery and Analysis (SIMPDA) (2015)

3. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer (2013)
4. Havur, G., Cabanillas, C., Mendling, J., Polleres, A.: Automated resource allocation in business processes with answer set programming. In: 11th International Workshop on Business Process Intelligence (2015)
5. Knublauch, H., Ryman, A.: Shapes Constraint Language (SHACL). Working Draft (work in progress), W3C (2016), https://www.w3.org/TR/shacl/

A Tool for the Analysis of DMN Decision Tables

Ülari Laurson and Fabrizio Maria Maggi

University of Tartu, Estonia
{ulaurson,f.m.maggi}@ut.ee

Abstract. The Decision Model and Notation (DMN) is a standard notation to specify decision logic in business applications. A central construct in DMN is a decision table. The rising use of DMN decision tables to capture and to automate everyday business decisions fuels the need to support analysis tasks on decision tables. This paper presents an open-source DMN editor to tackle three analysis tasks: detection of overlapping rules, detection of missing rules and simplification of decision tables via rule merging. The tool has been tested on large decision tables derived from a credit lending data-set.

Keywords. Decision Table, Decision Model and Notation, Camunda

1 Introduction

In 2014, the Object Management Group (OMG) published the Decision Model and Notation (DMN) standard [2]. The primary goal of this standard is to pro-

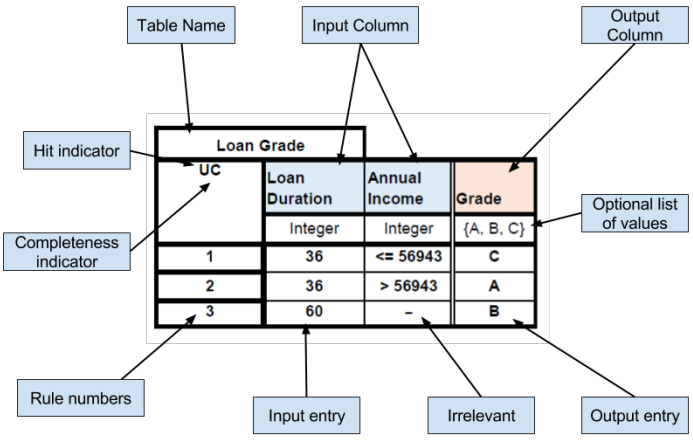


Fig. 1: Sample decision table with its elements

vide a common notation that is readily understandable for all business users including business managers, analysts, and developers. DMN consists of two levels: a "decision requirements level" and a "decision logic level". The decision requirements level defines input data needed to make a decision. The decision logic level describes how each decision is made. Decisions are usually expressed as decision tables. In decision tables, columns represent inputs and outputs of a decision, and rows represent rules. Columns typically are typed so that they have an associated domain. A rule is a conjunction of basic expressions (one basic expression per cell). Basic expressions are captured using a language known as S-Feel (Simplified Friendly Enough Expression Language).

In Figure 1, we see an example of a decision table. The rules in the decision table should be interpreted as follows:

1. If the customer asks for a loan duration of 36 and her annual income is less than or equal to 56943, then she gets grade "C."
2. If the customer asks for a loan duration of 36 and her annual income is greater than 56943, then she gets grade "A."
3. If the customer asks for a loan duration of 60, then she gets grade "B."

Using DMN decision tables for making critical business decisions raises the question of ensuring their correctness and simplicity to prevent costly defects. The tool presented in this paper implements scalable algorithms for two basic correctness checking tasks over DMN tables and one algorithm for simplification task [1]. The two correctness checking tasks are the detection of overlapping rules and the detection of missing rules. The detection of missing rules allows the user to check the completeness of the table, while the detection of overlapping rules allows the user to identify: (i) inconsistent rules, meaning two rules that overlap and that are associated with two different outputs; and (ii) redundant rules, meaning two rules that overlap and that have the same output.

2 Tool

We implemented our tool starting from `dmn-js`, the open-source rendering and editing toolkit of Camunda DMN.[2] We extended `dmn-js` to support correctness verification and simplification. Our tool can be found at `https://github.com/ulaurson/dmn-js` and a deployed version for testing is available at `http://dmn.cs.ut.ee/`. A screencast of the tool demo can be found at `https://www.dropbox.com/s/nc0vibw2jlv0p3l/DMN_Screencast.mp4?dl=0`.

The tool provides a syntactic check that verifies whether each cell of a decision table has the right content. The implemented algorithm checks if the content of each cell matches its column type.[3] If not, the cell becomes red and a tooltip is added. In Figure 2, we see two syntactic errors. In the third rule first column we have a syntactic error because this cell should contain a double but it contains a string. Another syntactic error is in the first rule second column. This cell should contain a string, but it contains an integer.

[2] `https://camunda.org/`

[3] Our tool supports types integer, string, Boolean, and double.

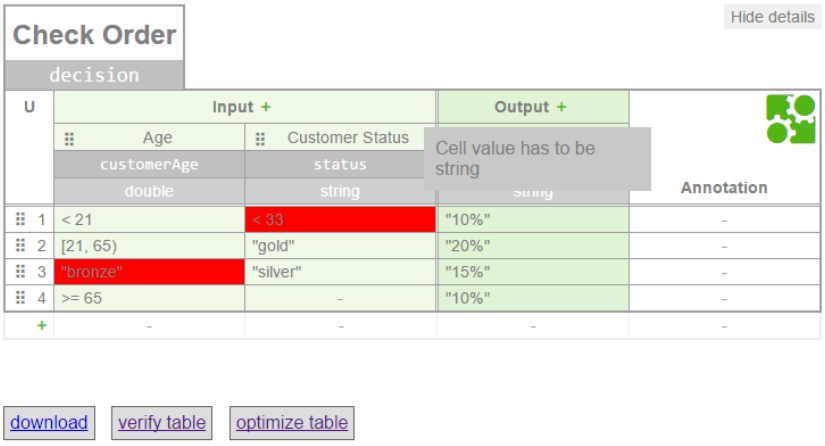


Fig. 2: dmn-js decision table with syntactic errors

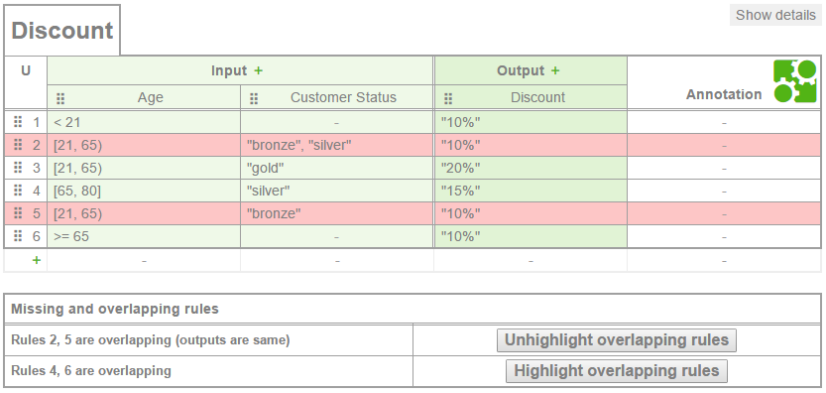


Fig. 3: dmn-js decision table with overlapping rules

When clicking verify table (see Figure 2), the tool finds all the overlaps in the decision table. The tool can find two types of overlaps: overlaps where inputs and outputs are the same and overlaps where inputs are the same and at least one output is different. In Figure 3, we see that the tool has found two overlaps. Rules 2 and 5, and rules 4 and 6 have overlaps. The overlapping rules are summarized in the "Missing and overlapping rules" table. This table contains maximal sets of overlapping rules with a non-empty intersection. If "(outputs are the same)" is explicitly mentioned in a set of overlapping rules, then these rules have the same inputs and outputs, otherwise they have different outputs. The overlapping rules are highlighted in light red.

When clicking verify table, the tool also finds the missing rules in the decision table. In Figure 4, the decision table has three missing rules. The first value inside

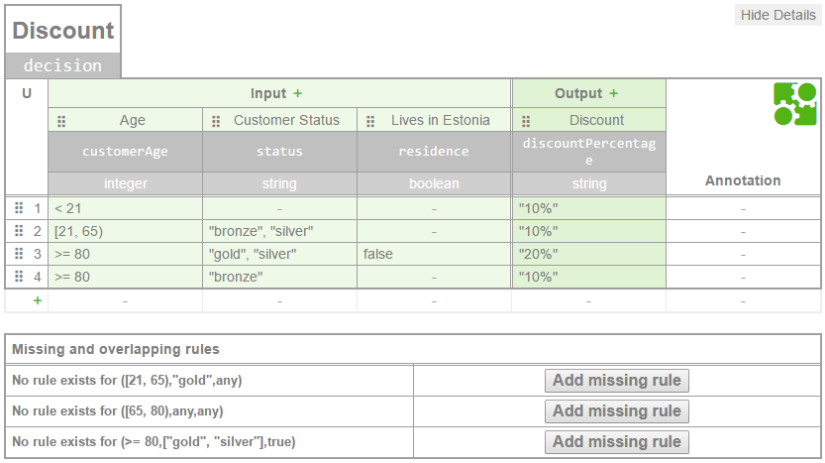


Fig. 4: dmn-js decision table with missing rules

the brackets shows what is missing in the first input column, the second value shows, what is missing in the second input column and so on. In Figure 4, we can see that there are three types of missing rules. In the first case, a categorical value is missing. Corresponding to interval [21, 65) there is no rule with "Customer Status" = "gold". In the second case, an interval is missing. In the decision table, there is no rule where "Age" = [65, 80). The third case has a missing Boolean value. With "Age" >= 80 and "Customer Status" = "gold" or "silver" there is no rule with "Lives in Estonia" = true. The missing rules are summarized in the "Missing and overlapping rules" table. The first column of this table contains information about the missing rules. The second column contains a button for adding the missing rules into the table.

When clicking simplify table (see Figure 2), the tool simplifies the decision table via rule merging. In Figure 5, the first decision table is a regular decision table that has no overlapping rules and no missing rules. The second decision table represents the simplified version of the first table. Rules 1, 2, and 6 can be merged because these rules only differ in one input (the outputs are the same). Rules 3, 4, and 7 can also be merged. Rules 1 and 3 in the second table cannot be merged because their outputs are different.

3 Maturity and Inherence

Based on the tool implementation, we have conducted empirical evaluations to compare the proposed algorithms with respect to existing approaches (in particular the one implemented in Signavio[4]) in terms of scalability, conciseness of the feedback provided to users (in case of overlapping and missing rules) and compactness of the simplified tables. The tests have been conducted on large decision tables derived from a credit lending data-set.

[4] http://www.signavio.com

Discount

Show details

U	Input +			Output +	Annotation
	Age	Customer Status	Lives in Estonia	Discount	
1	< 21	-	true	"10%"	-
2	[21, 80)	-	true	"10%"	-
3	[21, 80)	"gold", "silver"	false	"20%"	-
4	[21, 80)	"bronze"	false	"20%"	-
5	>= 80	-	false	"10%"	-
6	>= 80	-	true	"10%"	-
7	< 21	-	false	"20%"	-
+	-	-	-	-	-

Discount

Show details

U	Input +			Output +	Annotation
	Age	Customer Status	Lives in Estonia	Discount	
1	< 80	"bronze", "gold", "silver"	false	"20%"	-
2	-	"bronze", "gold", "silver"	true	"10%"	-
3	>= 80	"bronze", "gold", "silver"	false	"10%"	-
+	-	-	-	-	-

Fig. 5: Decision table before and after simplification

Execution times for missing rules detection are under 2 seconds. The detection of overlapping rules leads to higher execution times, due to the need to detect sets of overlapping rules and ensure maximality. In Signavio, if multiple rules have a joint intersection (e.g., rules {r1, r2, r3}) the output contains an overlap entry for the triplet {r1, r2, r3} but also for the pairs {r1, r2}, {r2, r3} and {r1, r3} (i.e., subsets of the overlapping set). Furthermore, in some cases, the overlap of pair {r1, r2} may be reported multiple times (and same for {r2, r3} and {r1, r3}). Meanwhile, our approach produces only maximal sets of overlapping rules with a non-empty intersection. Therefore, the number of sets of overlapping rules and the number of missing rules identified by our approach is drastically lower than the number of overlapping and missing rules identified by Signavio. For more information about our experimentation of the tool, the reader is referred to [1].

References

1. Calvanese, D., Dumas, M., Laurson, Ü., Maggi, F.M., Montali, M., Teinemaa, I.: Semantics and analysis of DMN decision tables. In Proceedings of the 14th International Conference on Business Process Management (BPM) 2016
2. Object Management Group: Decision Model and Notation (DMN) 1.0 (2015), `http://www.omg.org/spec/DMN/1.0/`

Agora - Speech-Act-Based Adaptive Case Management

Johannes Tenschert and Richard Lenz

Institute of Computer Science 6, University of Erlangen-Nuremberg,
{johannes.tenschert,richard.lenz}@fau.de

Abstract. Today's work is increasingly characterized by unpredictable collaborative processes called knowledge work. Some types of knowledge work are supported by case management tools which typically provide regulated access to case-related information. Knowledge workers are well aware of the pragmatic dimension of their communicative acts, but the systems they are using currently are not. This demo paper presents a prototype of a speech-act-based adaptive case management tool we call Agora. In Agora, we focus on interactions of a case and enable flexible documentation of ad-hoc interactions and activities as well as support for (semi-)structured processes. Interactions are linked to appropriate artifacts of the case. The approach enables useful inferences by the user and automatically by a system from all documented interactions and case-related data. For example, it can help in unveiling assertions and commitments, finding unfulfilled promises, and automatic reactions to certain interactions. Thereby, the approach facilitates integration of structured, semi-structured and ad-hoc processes.
Keywords: adaptive case management, speech act theory, knowledge-intensive business process

1 Background and Significance to BPM

Business process management is aimed at supporting cooperative work and shows its advantages especially in optimizing structured processes. However, in recent years the share of knowledge work has increased rapidly. Knowledge work involves the creation, distribution, or application of knowledge [2]. In contrast to a manual worker's clearly defined activities, knowledge work is characterized by abstractly defined tasks and the knowledge worker's responsibility for his own contribution in terms of quantity and quality [3]. In the US, around 50% of today's work is knowledge work [6]. Currently, interactions in knowledge work are supported by providing groupware and collaboration tools. Knowledge workers may also use an adaptive case management system (ACMS), tailored information systems, and other (process) support systems. Therefore, case data is scattered across many systems, and the overlapping structured, semi-structured

and ad-hoc processes involved in actual knowledge work further impede keeping track of related data, activities and interactions.

Our speech-act-based approach focuses on interactions of a case, and enables flexible documentation of ad-hoc interactions and activities, as well as support for (semi-)structured processes. It relates interactions to artifacts and makes the pragmatic intention of interactions explicit to facilitate useful inferences by the user or automatically by the system. Examples for these inferences are unveiling all assertions and commitments, finding unfulfilled promises, checking whether important stakeholders have been informed, or automatically reacting to certain interactions and situations, e. g. informing legal guardians about certain interactions. Moreover, the model enables compliance monitoring on interactions [9]. No process schema is necessary for a case, but if a more detailed schema can be given, the possibilities for support and inference increase. This demonstration introduces Agora, the prototype implementing our speech-act-based approach.

We use Speech Act Theory to classify and represent interactions. Speech Act Theory was first introduced by Austin [1], and further elaborated by Searle [4, 5]. Saying something is an action with a particular intention of the speaker. Not only utterances are speech acts, but rather all activities with the intention to send a message. Some types of interactions adhere to typical patterns, e. g. questions are usually followed by an answer. The speaker is well aware of this context and of his pragmatic intention, but the systems supporting him currently are not. A speech act consists of its illocutionary force F, i. e. the intention of the utterance, and its propositional content P. Searle's $F(P)$ framework allows that propositional content may also be a speech act, i. e. speech acts can be nested ("Alice informed me that bob promised...").

One example use case of Agora is writing a conference paper. Finding appropriate results for publication is not a structured process and varies from case to case. However, there are some similarities between cases: Usually, several authors are involved, case-specific agreements occur, and these agreements contain speech acts. By documenting and classifying these interactions, useful inferences and relations can be displayed. One could start by giving the project some working title and collecting artifacts (sketches, emails). The actual writing of the paper will be predominantly ad-hoc, but there are many interactions: questions, promises ("I'll write Section 2."), complaints ("I don't like the abstract!"), requests for meetings, and so forth. Some tasks could be automatized as a micro process [7], for example creating a git repository. This process may contain the creation of the repository, storing its location in the case's master data, and emails to stakeholders of the case to inform them about the repository. In most cases, the process of submitting a paper to a conference only differs slightly. A micro process might create some tasks to find deadlines and to document them, informing all authors, submitting the abstract, and submitting the paper. If it is accepted, more structured tasks are triggered: "Hooray" to all authors, submitting a camera ready version, registering at the conference, and booking the trip. At the conference, many contacts and interactions can be documented in Agora. After the trip, a structured process to settle travel expenses is triggered.

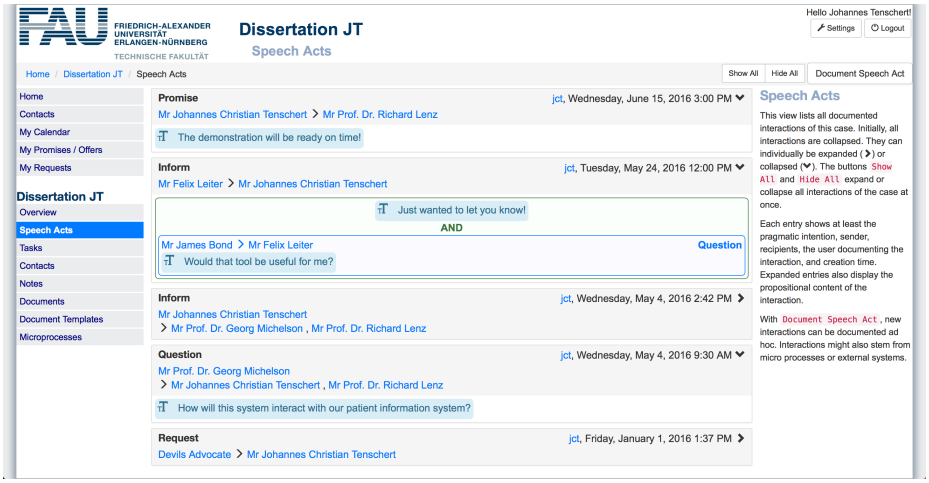


Fig. 1: Overview of speech acts of a case

Other examples that require handling cases in a flexible way and involve many interactions are treating and informing patients, handling legal cases, and supervising theses. Agora integrates well-known and well-established BPM techniques for structured processes with speech-act-based support for ad-hoc cooperation.

2 Model and Implementation

This section introduces the representation of cases in our approach, outlines the key features of the prototype Agora, and briefly describes its implementation.

2.1 Cases

Agora manages a set of cases. Each case consists of interactions, contacts, tasks, documents, notes, related process instances, and key-value-based master data. Interactions are documented speech acts with a specific creator of the documentation. Speech acts contain a sender, one or more recipients, an illocutionary force, and propositional content in the form of text, speech acts or production acts for artifacts, and logically connected text and acts adhering to Searle's F(P) framework. Speech acts can be linked to artifacts of a case, e. g. documents and tasks. Figure 1 shows how interactions of a case can be displayed. Here, users can expand and collapse propositional content of all or individual speech acts for an overview of all interactions of a case as well as for detailed information. Related process instances currently are restricted to micro processes [7]. In the near future, they will also contain structured processes in an external BPMS, e. g. to settle travel expenses or grade and properly archive a thesis.

2.2 Features

Agora provides flexible key-value annotations for cases and artifacts. These annotations can be organized in tabs and arranged freely according to the knowledge worker's preference. In order to make certain interactions involved in the generation of documents or completing tasks visible to the user and an inference engine, interactions are either automatically or manually linked to appropriate artifacts. Since Agora focuses on interactions, contacts involved in a case should be easy to maintain. Often, stakeholders are shared across many cases, e. g. attorneys and judges. Therefore, contacts are created once and can then be referenced. Contacts can be annotated with one or many roles in a case. The roles are not intended to be used like the traditional roles of BPM systems. They are intended to help managing the stakeholders involved, and *can* be used e. g. for simplifying document generation or preselecting contacts in a micro process.

Document templates may use certain artifacts (e. g. contacts) and master data (e. g. file reference, working title) of a case to generate word documents or emails. For example, while dealing with a thesis and if the student is a documented stakeholder of the case, the registration at the exam office can be generated. Also, travel expense settlement forms for attending a conference could be pre-populated. Micro processes [7] are tightly integrated into the prototype to automate routine fragments. They allow parallelism, missing attributes of interactions and actions that are clarified during execution, and flexible user input for several tokens of a process instance at the same time. Inferences currently are hard-coded into the system, but already allow for example to find unfulfilled promises and to display all assertions. The system is prepared to provide automatic reactions for interactions according to the type of interaction and the case's master data, e. g. to inform legal guardians about activities. The Agora client is a single-page web application. It synchronizes certain artifacts (e. g. contacts, cases, current view) continuously to facilitate collaboration.

2.3 Architecture and implementation

The architecture of our approach is described in [8]. On the server side, Agora is implemented in Java servlets. The servlets provide REST interfaces that return JSON. Clients access these REST interfaces with a single-page web application based on HTML5, AngularJS, and Bootstrap. Case data and processes are stored in a relational database and accessed with Hibernate.

Moreover, a JSON library has been developed to easily map objects to different profiles, i. e. to not reimplement the same classes for specific views (confidentiality, volume). An additional layer for templates of word documents has been created based on docx4j. This document generation layer handles merge fields, parameters, and dynamically generated formatted paragraphs and runs.

3 Maturity and Future Work

The Agora prototype is intended to demonstrate how interaction artifacts and the pragmatic intention of interactions can be integrated into an adaptive case

management system. It has not yet been evaluated with end users. Agora supports interactions that are part of structured processes, emerge in semi-structured processes, or are documented in an ad-hoc fashion. The system provides a flexible data model that allows knowledge workers to create arbitrary annotations for cases and artifacts, and to relate interactions to artifacts. A screencast is available at `http://www6.cs.fau.de/people/johannes-tenschert/bpm-2016`.

Future versions of Agora will improve inferences, usability, and traceability of interactions. Currently inferences are hard-coded. There is active development to integrate business rules according to [8, 9] for integration of structured, semi-structured and ad-hoc processes as well as for compliance monitoring. It is intended to allow user-definable domain-specific business rules. The data model of artifacts in Agora is flexible, and the model of artifacts typically managed with a smartphone and groupware (e. g. contacts, tasks, and notes) is very similar to the appropriate web standards vCard and iCalender in order to facilitate synchronization in the future. Therefore, knowledge workers can choose their preferred gadgets and tools to work on a case. Finally, the model and implementation will be improved to further support traceability of interactions. Currently, a case is one conversation, and the interactions forming logical connections, e. g. accepting a specific proposal, are not explicitly linked together. Ideally, finding the reason of performing certain (speech) acts should not require to read the whole conversation, and typical patterns of interaction could be supported without extensive manual documentation.

The vision for a final prototype is that even though case data may still be distributed across certain devices and BPMSs, the ACMS would be a system of record that includes and references all (interaction) artifacts of a case. It should suggest appropriate activities, integrate with the tools knowledge workers actually use, reduce the time to hunt for information and creating routine artifacts, and provide useful inferences for dealing with the case at hand.

References

1. Austin, J.L.: How to do things with words. Oxford university press (1975)
2. Davenport, T.H.: Thinking for a Living: How to Get Better Performances and Results from Knowledge Workers, chap. What's a Knowledge Worker, Anyway?, pp. 1–24. Harvard Business Press (2005)
3. Drucker, P.F.: Knowledge-worker productivity: The biggest challenge. California Management Review 41(2), 79–94 (1999)
4. Searle, J.R.: Speech acts: An essay in the philosophy of language. Cambridge university press (1969)
5. Searle, J.R., Vanderveken, D.: Foundations of illocutionary logic. Cambridge University Press (1985)
6. Swenson, K.D.: Robots don't innovate - innovation vs automation in BPM (May 2015)
7. Tenschert, J., Lenz, R.: Supporting knowledge work by speech-act based templates for micro processes. In: AdaptiveCM 2015–4th International Workshop on Adaptive Case Management and other non-workflow approaches to BPM (2015)

8. Tenschert, J., Lenz, R.: Towards speech-act-based adaptive case management. In: AdaptiveCM 2016–5th International Workshop on Adaptive Case Management and other non-workflow approaches to BPM (2016)
9. Tenschert, J., Michelson, G., Lenz, R.: Towards speech-act-based compliance. In: 2016 IEEE 18th Conference on Business Informatics (2016)

Unicorn meets Chimera: Integrating External Events into Case Management

Jonas Beyer, Patrick Kuhn, Marcin Hewelt, Sankalita Mandal, Mathias Weske

Hasso Plattner Institute, University of Potsdam, Germany
{Marcin.Hewelt,Sankalita.Mandal,Mathias.Weske}@hpi.de

Abstract. Case management allows knowledge workers to model and enact flexible, knowledge-intensive business processes. Such processes occur in many domains, e.g. logistics or healthcare, and the exact course of a case can not be pre-specified, because it heavily depends on case data, user decisions, and external events, which take place during runtime. This work extends our case management engine Chimera with the capability to incorporate external events. To this end we integrate Chimera with the event processing platform Unicorn, with the result that external events can now trigger new cases, provide case data, or abort activities. This demo is aimed at practitioners and academics in the field of flexible business processes and case management.

Keywords: Case Management, Business Process Management, Complex Event Processing, Case Execution, flexible Business Processes.

1 Overview

Case Management is an approach suitable for the modeling and execution of knowledge-intensive business processes that center around a case. When executing a case, competent knowledge workers try to achieve the specific case goal, by aligning their activities with the emergent requirements of the case. However, the sequence of activities executed towards the goal, depends on the specific circumstances of the case, which only become apparent during case execution, and hence can not be pre-specified.

When dealing with highly flexible processes, integrating external events is necessary to enable case workers to react efficiently and adapt their work to the current situation. Complex event processing is an already proven mean to provide high level events relevant to the course of the process [1], eventually cases. While process modeling languages like BPMN 2 [5] allow to model several kinds of events, process engines like Camunda or Activiti[1] are limited to their engine-internal events.

[1] see `http://camunda.de` resp. `http://activiti.org`

This work builds on the case management engine Chimera[2] that was presented at last year's BPM demo under the name JEngine [2]. Here we extend Chimera with capabilities to deal with external events by registering event queries with the event processing platform Unicorn[3], and reacting on received event notifications. Furthermore, we present the case modeling tool Gryphon. It allows knowledge workers to model event types as part of the domain model of a case model, as well as process fragments that are annotated with event queries.

2 Fragment-based Case Management

In our project, we consider the fragment-based case management approach (fCM) by [3, 4], where a business scenario is represented by a *case model* consisting of a set of fragments, a domain model, and a set of life cycles. Each fragment is a process model that contains control flow necessary to describe how to handle a subsection of the case. Upon execution, the fragment instances are dynamically combined based on their data dependencies. The data classes and their associations, as well as their domain specific attributes are defined in the domain model. Each data class has a corresponding object life cycle that specifies possible state transitions a data object of that class can undergo during case execution. The Chimera approach uses BPMN for the fragments, UML for the domain model, and state transition graphs for the life cycles. The semantics of fragment-based case management has been formally defined in [3].

3 Event Integration into fCM

We explain the concept of integrating events with the usecase of asparagus harvest. External factors like temperature and weather prediction influence the harvesting process. One important aspect of the usecase is depicted in Fig. 1 that shows one of several process fragments contained in the case model of our usecase. In order to balance supply and demand for asparagus the harvest date needs to be coordinated with other asparagus farmers in the region.

The fragment in Fig. 1 is enabled once the `Harvest Plan` has been `[created]`. The farmer then starts preparing resources and equipment for the harvest. But as depicted by the event-based-gateway, if the `Market price dropped` event occurs before the harvest date, the `Market Situation` data object is updated and the harvest has to be postponed.

In the remaining part of the section, the work flow for implementing the use case in our execution environment is discussed. Figure 2 provides an overview of all components and their interactions.

[2] see `https://bpt.hpi.uni-potsdam.de/Public/ChimeraDoc`
[3] see `http://bpt.hpi.uni-potsdam.de/UNICORN`

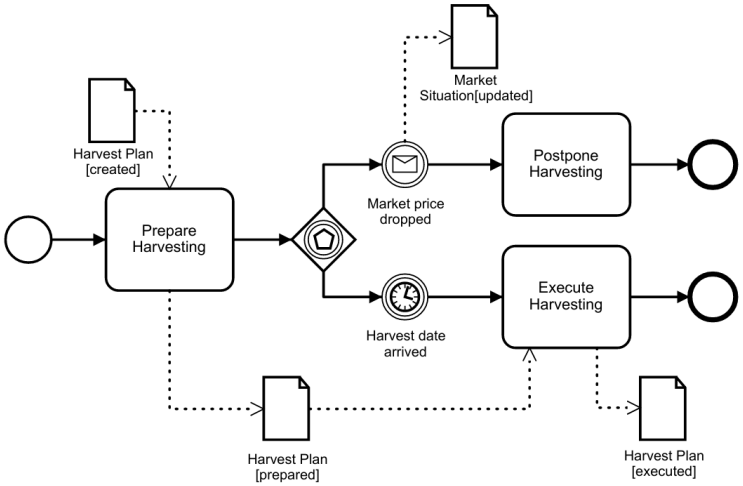


Fig. 1: A process fragment from the case model

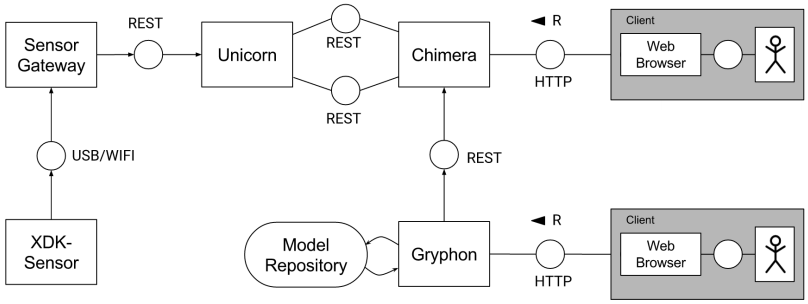
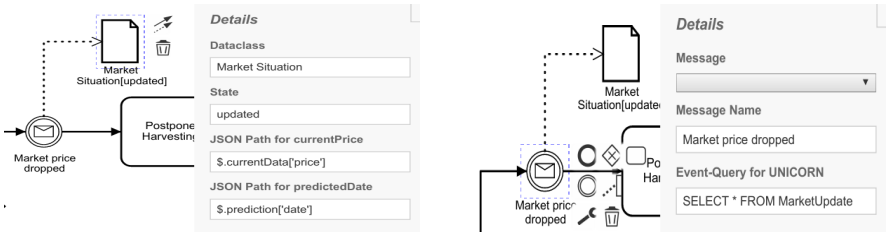


Fig. 2: Architecture

Case model creation. In order to include external events in the execution of a case, they first have to be modelled. It is possible to define event types that describe the domain and the attributes of events. Specific events triggering actions in the case can be modelled using event queries. Event queries are similar to database queries, the difference being that they operate on event streams instead of persistent data. If an event from the stream matches the query, the event is then sent to the execution engine. We chose the Event Processing Language (EPL) provided by Esper[4], as language to express those event queries. We decided to reuse the catching message event to model enabled events as notifications can be considered as receiving messages.

To model the case, we used Gryphon, a web-based tool built around bpmn.io[5] based on a node.js[6]-stack. Gryphon allows to model process fragments, domain model, life cycles, and termination conditions. The completed case model can then be deployed to a running Chimera instance.

[4] see `http://www.espertech.com/products/esper.php`
[5] An open-source BPMN modeler implemented in Javascript `http://bpmn.io`
[6] see `http://node.js`



(a) Modeling of data attribute bindings (b) Modeling of event queries

Fig. 3: Modeling extensions in Gryphon

Model deployment. Chimera is an engine for executing case models, consisting of a web-based frontend and a backend communicating via a RESTful API. Chimera parses the received case model and registers event types and case start queries with Unicorn. The case can then be executed through the web UI, where the user is presented with an overview of all case models.

Chimera also supports incorporating events as case data. Events are received in the form of JSON objects, where each attribute field has a specific value. Thus, we can parse the JSON and set data object attributes according to the event attribute values. This is implemented by evaluating JsonPath[7] expressions. The user can specify one JsonPath expression per attribute of the data object that is used to persist the values, as seen in figure 3a.

Event and query registration. Whenever an event is reached during execution in Chimera, an event query is registered with Unicorn. Unicorn is an event processing platform built around the Esper Event Processing Engine that allows to manage event types, event queries, and notifications both via a web-based UI and a REST API. Events are sent to Unicorn either via a REST API or by means of adapters, that periodically call webservices.

A general overview of the event registration process is shown in Fig. 4. The registered event queries can be divided into two groups. Case start queries have to be registered when a new case model is deployed to Chimera and remain registered until the case is deleted. All other event queries are registered as soon as the respective event control node is reached. The annotation to register event queries has been shown in Fig. 3b. Each query is registered with a specific id, which is used to correlate the event control node to the event query after it was triggered. The queries are unregistered from Unicorn when the event is triggered or skipped.

Event generation. In our use case, events were produced by a sensor unit. Because we did not have access to real "production fields", the sensor unit used was the Bosch XDK developer kit[8], a package with multiple integrated sensors for prototyping of IoT applications. The unit sends measurement values over

[7] see `http://goessner.net/articles/JsonPath/`
[8] see `http://xdk.bosch-connectivity.com`

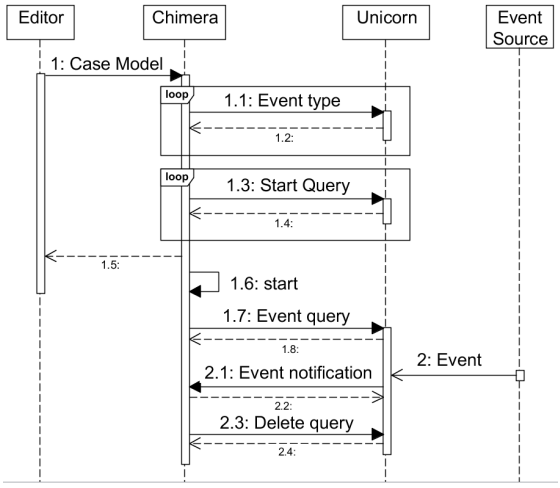


Fig. 4: Event integration sequence

wireless network to a gateway that parses the proprietary format of the received data and forwards it to Unicorn using the REST API.

Reactions to events. If the event is linked to a case start query, the case is initiated. The specified data objects are created using the information given by the start query — this includes the initial data object state and its attribute values. Otherwise, the event control node associated to the query id is retrieved and executed. If the control node has outgoing data objects that define data bindings, the event data is evaluated with the help of the JsonPath expressions, and the attribute values are saved.

4 Conclusion

In this paper, we presented our prototypical adaption of a case management execution engine to handle real life events sent by a sensor. To this end, we adapted the modeling component Gryphon to allow event modeling and integrated the event processing platform Unicorn and a Sensor Gateway into the architecture. The case engine itself had to be adapted to parse and register event types and event queries, as well as to react to received event notifications.

This contribution is part of an ongoing project to develop a highly usable environment for knowledge workers to model and enact fragment-based case models. Source code, documentation, and screencast of the Chimera case engine are available at `https://bpt-lab.org/fcm`.

References

1. O. Etzion and P. Niblett. *Event Processing in Action.* Manning Publications, 2010.

2. S. Haarmann, N. Podlesny, M. Hewelt, A. Meyer, and M. Weske. Production case management: A prototypical process engine to execute flexible business processes. In *Proceedings of the BPM Demo Session*, pages 110–114, 2015.
3. M. Hewelt and M. Weske. A Hybrid Approach for Flexible Case Modeling and Execution. In *BPM Forum*, LNBIP. Springer, 2016. (accepted for publication).
4. A. Meyer, N. Herzberg, F. Puhlmann, and M. Weske. Implementation framework for production case management: Modeling and execution. In *Enterprise Distributed Object Computing (EDOC)*. IEEE, 2014.
5. Object Management Group. Business Process Model and Notation (BPMN), Version 2.0.2, 2013.

Composite State Machine Miner: Discovering and Exploring Multi-perspective Processes

Maikel L. van Eck*, Natalia Sidorova, and Wil M.P. van der Aalst

Eindhoven University of Technology, The Netherlands
{m.l.v.eck,n.sidorova,w.m.p.v.d.aalst}@tue.nl

Abstract. Process mining provides fact-based insights into processes based on behaviour captured in event data. An important aspect of this is the discovery of process models from such data. Traditionally, the focus of process discovery is on learning the ordering of activities. We deviate from this dominating activity view on processes to focus on states and state changes. Specifically, we aim to discover state-based models for processes where different facets, or perspectives, of the process can be identified. In this paper we describe an interactive process discovery tool that can be used to discover and explore state-based models for such multi-perspective processes: the *Composite State Machine Miner*. It quantifies and visualises the interactions between perspectives to provide additional process insights. This tool has been used to analyse the BPI Challenge 2012 data of a loan application process and product user behaviour data gathered by Philips during the development of a smart baby bottle equipped with various sensors.

Keywords: process discovery, state-based models, multi-perspective processes, interactive process exploration

1 Introduction

We assume that the reader is familiar with the basic concepts of process mining and process discovery, and we refer to [1] for an in-depth overview. The goal of most process discovery approaches is to obtain models that describe the ordering of activities within a process. These approaches usually have an implicit notion of the state of a process.

In this work we deviate from the dominating activity view on processes in order to focus explicitly on process states and state changes. This state view is more intuitive than an activity view for processes for which state information is explicitly available. Examples of such explicit state information are the diagnosis

* This research was performed in the context of the IMPULS collaboration project of Eindhoven University of Technology and Philips: "Mine your own body".

of a patient in a healthcare process or the status of an order in a purchasing process.

When studying the states of a process, that single process can have different facets, or *perspectives*, each with their own state space. For example, consider the homeostatic process in a person, parts of which regulate sleep and nutrition. From the perspective of sleep the state of a person can be e.g. awake or asleep, while the state of the nutrition perspective can be e.g. hungry, eating or sated. The state of a person at any point in time is the composition of the state of both perspectives. These perspectives have individual process cycles, but there are interdependencies between states from different perspectives, e.g. people are awake while eating. Our goal is to analyse these interdependencies between perspectives for multi-perspective state-based processes.

An approach to achieve this is described in detail in [3]. In this work we discuss the implementation of this approach: the *Composite State Machine (CSM) Miner*.

2 Implementation

The CSM Miner has been implemented as a plug-in of the ProM framework [4]. It is obtained by installing ProM 6.6 or later from `http://www.promtools.org/` and then using the ProM package manager to instal the CSMMiner package. Example logs that can be used with the CSM Miner can be obtained from `https://svn.win.tue.nl/repos/prom/Packages/CSMMiner/Logs`. There is a screencast providing a demonstration of the main features of the miner at: `https://svn.win.tue.nl/repos/prom/Packages/CSMMiner/Documentation`.

The main assumption behind the CSM Miner is that each state of the multi-perspective composite process under study is a vector of the states of its perspectives. That is, for a process with n perspectives each state of the composite process is a state of the form: $s = (s_1 \times \ldots \times s_n)$. The CSM Miner discovers a state machine describing the states and state changes of the composite process, as well as a state machine for each perspective. Transitions $(s \rightarrow s')$ in the discovered composite state machine represent state changes in one or more perspectives, while transitions $(s_i \rightarrow s'_i)$ in the discovered state machine of perspective i represent a change in that specific perspective.

The input data for the CSM Miner is assumed to be an XES event log [4] where each event represents a state change in a specific perspective. In Fig. 1 an example of the desired input is shown. Each trace in the log is required to contain attributes of the form `process:initialstate:[perspectiveName]` specifying the initial state of each perspective at the start of the trace. Each event is required to contain an attribute of the form `process:name` specifying the perspective for which it is a state change.

An example of a composite state machine and its perspective state machines discovered by the CSM Miner is shown in Fig. 3. Each state machine is displayed separately in an interactive visualisation where states and transitions can be dragged to move them around.

```
<log xes.version="1.0" xes.features="nested-attributes" openxes.version="1.0RC7">
    <string key="concept:name" value="Example Log"/>
    <trace>
        <string key="concept:name" value="Day1"/>
        <string key="process:initialstate:Feeding" value="Hungry"/>
        <string key="process:initialstate:Sleeping" value="Awake"/>
        <event>
            <string key="process:name" value="Feeding"/>
            <string key="concept:name" value="Eating"/>
            <string key="lifecycle:transition" value="start"/>
            <date key="time:timestamp" value="1970-01-01T06:00:00.000+01:00"/>
        </event>
        <event>
            <string key="process:name" value="Feeding"/>
            <string key="concept:name" value="Sated"/>
            <string key="lifecycle:transition" value="start"/>
            <date key="time:timestamp" value="1970-01-01T06:15:00.000+01:00"/>
        </event>
```

Fig. 1: A partial XES event log that can be used as input for the CSM Miner. Each trace in the log is required to be annotated with attributes specifying the initial state of each perspective and each event is required to be annotated with the perspective to which they belong.

States and transitions are annotated with statistics and additional statistics are shown at the bottom of the visualisation for the state or transition that is currently selected. For each state the statistics show the total number of times this state was observed to occur in the log, as well as the total time that was spent in this state. For each transition the statistics also show the total number of times this transition was observed. The percentage for each transition shows the observed frequency with which this transition was taken from the source state of the transition, i.e. if a state has only one outgoing transition then it is annotated with 100% and if there are two outgoing transitions that have been observed equally many times then they are each annotated with 50%.

By clicking on a state or transition the states and transitions in the other models that co-occur with the selected element are highlighted. The highlighted states also show two different statistics: confidence and lift. Confidence expresses what percentage of time the highlighted state was observed together with a selected state compared to the total time spent in that selected state. Lift expresses how much the confidence differs from the percentage of time that a highlighted state is expected to occur independent of the selected state. For a more detailed discussion of confidence and lift we refer to [3].

The visualisation of the discovered models also contains functionality that enables the user to transform the model. This functionality can be accessed from an expandable menu, as shown in Fig. 2, and allows the user to remove selected states from the model and to aggregate selected states. During the model transformation the statistics are recalculated.

3 Case Studies

The Composite State Machine Miner has been used during the analysis of two case studies. The first case study concerned the BPI Challenge 2012 data of

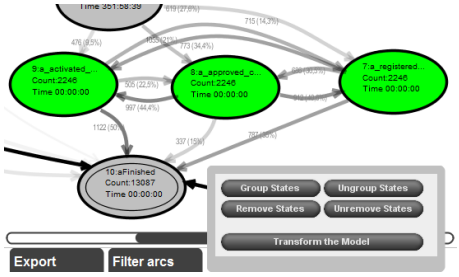


Fig. 2: The green states are selected for aggregation by the user because they are always executed together, but in arbitrary order.

a loan application process [2]. The second case study involved the analysis of product user behaviour data gathered by Philips during the development of a smart baby bottle equipped with various sensors. These case studies and the insights provided by the CSM Miner are discussed in detail in [3].

The models shown in Fig. 3 have been discovered on the BPI Challenge 2012 data. These models are much more structured than models discovered by traditional process discovery algorithms on the same data. Analysing them provided useful insights into the difference between automatically and manually processed applications and the effectiveness of fraud investigation.

4 Conclusion

In this paper we have presented the Composite State Machine Miner, an interactive tool for the discovery and exploration of state-based multi-perspective processes. The CSM Miner can be used to study the interactions between perspectives in such processes. This has been used successfully in two case studies.

Future work that we plan to do is focussed at improving the practical usability of the tool. For example, it would be useful to suggest states or relations between perspectives that are likely to be of interest to the user, to avoid having to click on every element of the discovered models to explore the results. Also, additional support for different types of relations between perspectives would enable the CSM Miner to generate additional insights.

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. van Dongen, B.F.: BPI Challenge 2012 (2012), http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
3. van Eck, M.L., Sidorova, N., van der Aalst, W.M.P.: Discovering and Exploring State-based Models for Multi-perspective Processes. In: 14th International Conference on Business Process Management (BPM). p. In press. Springer (2016)
4. Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: XES, XESame, and ProM 6. In: Information Systems Evolution, pp. 60–75 (2011)

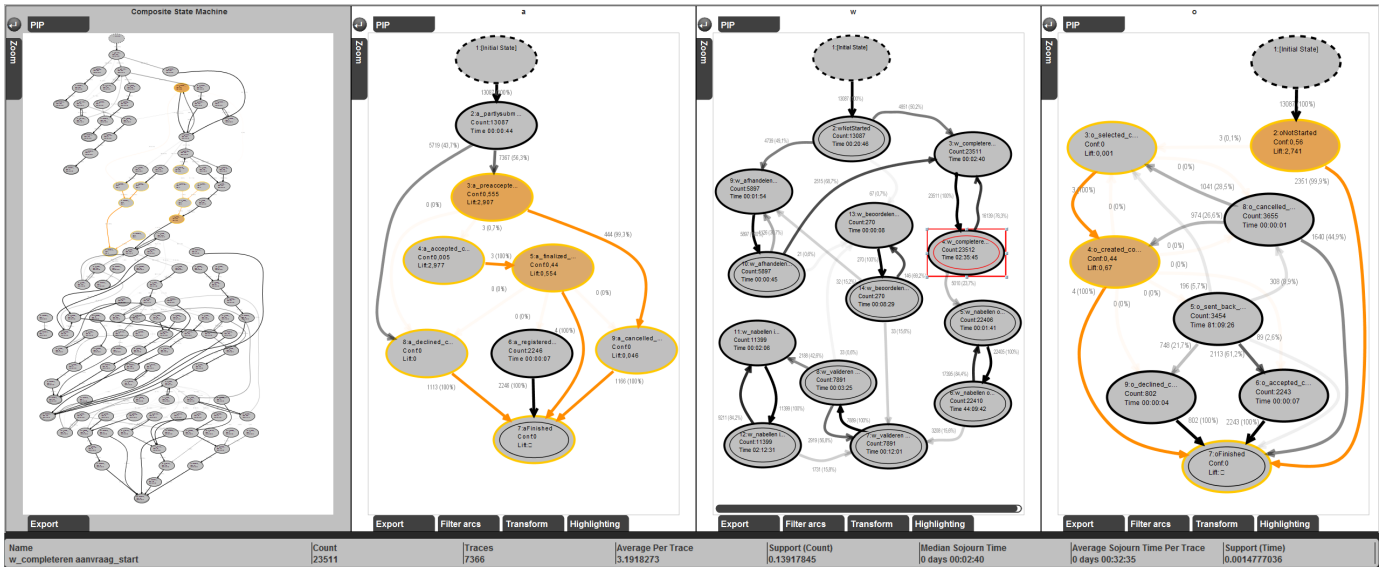


Fig. 3: The interactive visualisation of a discovered composite state machine. The selected state is denoted with a red box and its co-occurring states and transitions are highlighted in the other perspectives and the overall view based on their *confidence*. The statistics below the visualisation refer to the selected state or transition.