VisionWaves: Aligning Business Process Management and Performance Management to Achieve Business (Process) Excellence

Marc Kerremans
m.kerremans@visionwaves.com

Abstract. Today's economic climate means businesses need to be as effective and efficient as possible, and to make the smartest possible decisions. The way to achieve this is by integrating Performance Management (PM) and Business Process Management (BPM) – a combination that is a prerequisite of Intelligent Business Operations. BPM on its own is not enough, omitting the context of processes. By adding PM you can achieve closed loop performance management, where metrics are compared with business objectives and the results fed back to improve processes and decisions. VisionWaves brings a business model driven approach to the table coping with these requirements and allowing Visual and Connected Management. Client experience confirms that PM and BPM are far more powerful when integrated together than individually.

1 Gaining competitive advantage through better processes and decisions and the limits of BPM

Given the economic and institutional crises in our globalized economy, the way to compete is to run your business with maximum efficiency and effectiveness, and to make the smartest possible decisions. One of the approaches is to focus on "the integrated business processes and capabilities that together serve customers in ways that are differentiated from competitors and that create the organization's formula for business success" [1]. Does BPM on its own deliver this right kind of efficiency? Does the balance between efficiency and agility meet business needs? Is the level of intimacy appropriate for every customer? Let's look at some examples:

- A customer delivery process can be very efficient (for example, following a Lean program) – in fact, it may not contain any idle time at all. But efficiency is not always matched with effectiveness. For example, it may be that because of marketing campaigns in some regions, there are configuration problems and stock breakdowns.
- By definition, agility can necessitate some inefficiencies. You need more than minimal stock levels to keep a manufacturing conveyor belt running; similarly, economic growth arguably requires some level of frictional unemployment.
- A call center representative may be confronted with a demanding customer, and be forced to decide whether to prioritize overall efficiency or service to this one customer. Does the agent mark the customer as a lead, get off the

phone, and move on to the next customer, or continue the conversation with this customer and let others wait? An optimized process won't help the agent decide, unless the process also provides information about the individual customer's value to the business.

2 Combining BPM and PM

To yield the desired competitive advantage, structured interoperability is required across the organization's entire BPM, PM and application environments. This interoperability must be business-driven, which means that the initiative needs to start from a broader context than the processes themselves: that of the business architecture, business model, or value chain.

The alignment of BPM and PM can be seen from two main perspectives:

- Injecting PM into business processes to improve decision-making – i.e. taking an inside-out view of the of the decision
- Closed loop performance management – i.e. taking an outside-in view of the decision

2.1 Injecting PM into business processes to improve decision-making

Business rules are often used to inject PM into BPM and hence provide decision support. This is a valid approach that makes processes more flexible, offers additional analytic capabilities, and hides much of the complexity that is typical of PM.

However, rules engines alone are not enough. They cannot cope with the typical scenario of proliferating business models, products, services, channels, customer segments, and value expectations from different stakeholders. Nor do they provide the vital ability to understand changing environments and respond to that understanding, especially when we look at changing sets of goals.

By combining PM with the rule-based approach associated with BPM, you can monitor the decision-making process from an analytical perspective and adapt readily to changing goals. Errors in business rules will be detected sooner, and rules can be changed as necessary, either automatically or manually.

2.2 Closed loop performance management

Most BPM initiatives capture metrics to assess the efficiency of a process. A few initiatives have shown, though, that more can be done by setting process metrics in the broader context of a value chain or a business model. It then becomes possible to understand the impact of a given process on the overall performance of the business: either on its overall strategic objectives or a specific business campaign. That understanding can lead to better decision-making at an organizational level.

The results of performance monitoring become even more valuable if they are used to adjust business processes and objectives. This closed loop performance management often involves human intervention to improve the way decisions are

made – for example, the call center agent might be instructed to look at a metric of the customer's value to the company before deciding how long to spend on the call.

3 The VisionWaves proposition

VisionWaves brings a business model driven approach to the table coping with these requirements and allowing visual and connected management.

3.1 Visual Management

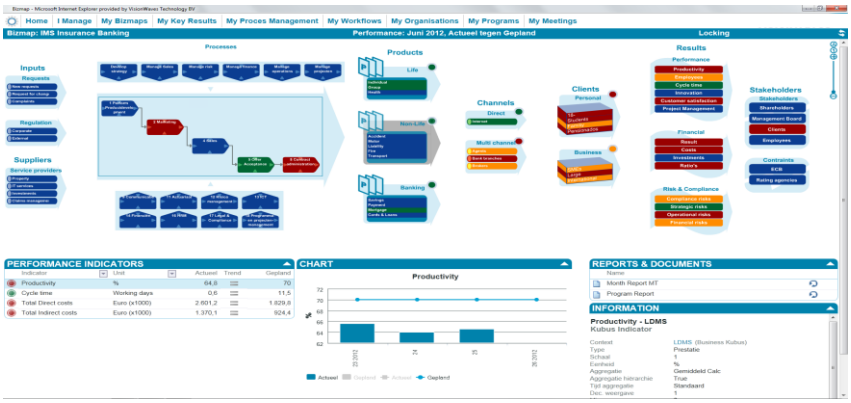

Fig. 1. Example of an implemented instance of a business model.

Core to the offering is the VisionWaves business model methodology that delivers a visual representation of the coherence and dependencies between the constituents of the business through depicting an easy to understand business model or value creation model. This visual insight that is missing in most alternatives is one of the key differentiators of VisionWaves within the enterprise performance management market. Furthermore this business model is dynamic meaning that it is context aware and is capable to continuously interact with its environment and its users.

3.2 Connected Management

VisionWaves delivers full visibility of value creation, performance, processes and risks in integrated, strategically aligned and actionable management cockpits that are role based and that are generated and maintained by the intelligent framework itself. Based on the same underlying data everybody will get a role based portal that reflects actual, executable, connected, and integrated data.

3.3 Model Driven Application Framework

VisionWaves starts with the representation of the 'business model' including customer value, the different distribution channels through which this value is delivered, products, services, processes, organization, roles, suppliers, contracts and risks. This meta-model is stored into an object repository and therefore delivers a model-driven object model.

Next step is the connection of 'meta' performance indicators to the elements of the meta-model that are also stored in the same object repository.

Furthermore to feed real data into the object repository a meta-data model is configured as well as the (meta) description of how these data are loaded into the same object repository.

Finally even the presentation layer (dashboards, cockpits) is created and maintained by the framework.

Perhaps the biggest contribution of this model driven application framework is the way it handles changes. When there is a change in the external environment or internal context management objectives, controls and of course reporting has to be realigned. In this model driven approach this can be done by reconfiguring the models or any of the model components in real-time, followed by the immediate adaptation of all related cockpits, dashboards, and reports.

4 Client case study: Bank achieves Operational Intelligence by combining BPM and PM

4.1 Situation after BPM-only

Previous BPM initiatives resulted in leaner operations and well-documented processes, but essential business elements were still missing. For example:

- The impact of the change on business results or business value was not clear
- There was a lack of information to support senior management decisions, and the impact of those decisions on operational execution was hard to establish
- There was not enough information about current processes for the COO to know whether a proposed action, such as a new market campaign, was viable in a particular region
- Performance at each level of the management hierarchy (COO, value chain owner, process owner, team owner) was measured, but it was not possible to see how one level impacted another

4.2 Results after implementing VisionWaves

- All management levels, from COO to operational team leaders, now have performance information about processes, customers, finance and capabilities to support their decisions
- The impact of performance at one level on another level can be seen, and there is a daily "performance dialogue" between all hierarchical levels
- This makes it possible to work together to achieve corporate objectives
- A range of information to support business campaigns is now available – it is easy to assess whether they are viable, and then measure their impact
- Through appropriate use of performance feedback, the bank has closed the loop between initiating actions, monitoring performance and taking new actions

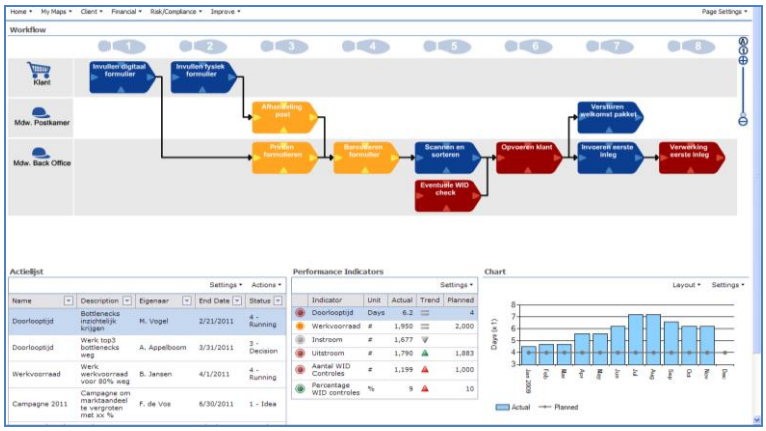

Fig. 2. Example of a Process Manager Cockpit.

5 Conclusion

Combining BPM and PM allows management and staff to make better and timelier decisions and the organization becomes more efficient and effective. This will help achieving business (process) excellence and is a crucial step for any enterprise with its sights set on Operational Intelligence or Intelligent Business Operations.

References

1. Thomas H. Davenport, Jeanne G. Harris, Competing on Analytics, (Harvard Business School Press, 2007), p. 187

CPN Tools 4: A Process Modeling Tool Combining Declarative and Imperative Paradigms

Michael Westergaard[1,2⋆] and Tijs Slaats[3,4⋆⋆]

[1] Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands
[2] National Research University Higher School of Economics,
Moscow, 101000, Russia
[3] IT University of Copenhagen
Rued Langgaardsvej 7, 2300 Copenhagen, Denmark
[4] Exformatics A/S, Lautrupsgade 13, 2100 Copenhagen, Denmark
m.westergaard@tue.nl, tslaats@itu.dk

Abstract. CPN Tools is a tool for modeling, simulating, and analyzing colored Petri nets. The latest iteration of the tool, CPN Tools 4, extends this with constraints known from declarative languages such as Declare and DCR Graphs. Furthermore, this version introduces an explicit process perspective, powerful extensibility allowing third parties to extend the tools capabilities, and a visualization perspective making it possible to make high-level visualizations of executions directly in the tool.
In our demonstration, we show how it is possible to create models incorporating declarative and imperative constructs and how to use these models to generate simulation logs that can be directly imported into ProM. We show off the new process perspective on top of colored Petri nets, exemplify the use of the perspective to generate readable Java code directly from models, and show how the visualization perspective makes it possible to show the formal underlying model alongside an easier-to-grasp for non-experts high-level visualization.
Our intended audience comprise current users of CPN Tools interested in recent developments and practitioners interested in colored Petri nets and hybrid models. We expect to tailor each demonstration to the wishes of the audience.

Standard imperative languages are suitable for the description of well-structured and well-understood processes. On the other hand, processes that are less well-understood or less well-structured, are often easier modeled using a declarative approach, where instead of specifying the next task to execute, constraints between tasks are described. Popular languages for imperative specifications include BPMN and (colored) Petri nets. Declarative modeling is a more recent

⋆ Support from the Basic Research Program of the National Research University Higher School of Economics is gratefully acknowledged.
⋆⋆ This research is supported by the Danish Agency for Science, Technology and Innovation through an industrial PhD Grant.

and less matured approach which has so far not found widespread application in industry yet, however the two declarative languages Declare [6] and DCR Graphs [2, 3] have been studied extensively in academia over the last decade. Declarative languages do not explicitly specify flow of control, but instead specifies constraints between actions; examples of such constraints are init(A), meaning that any execution has to start by executing A, and response(A, B), meaning that after executing A, B has to be executed at some point. Other constraints deal with choices and variations of the response constraint.

Hybrid modeling. Recently interest has emerged in hybrid approaches, where some aspects of a process are specified directly using imperative constructs and other aspects declaratively. This is useful if part of the process is well-structured and part is more free, or for going from an abstract, little-understood process, often modeled more naturally using declarative constraints, to a more concrete implementation which by nature is often more imperative. One such hybrid approach is implemented in CPN Tools 4 [5, 7]. This approach combines the places and transitions of colored Petri nets with the constraints of the Declare and DCR Graphs languages. Fig. 1 shows an example of a mixed declarative and imperative model. In the upper half of the screen we describe the registration of a patient using an electronic patient record, which is basically form-filling and well-suited for an imperative approach. In the bottom half we describe the treatment of the patient which is strongly knowledge-based, therefore more flexible and hence modeled using a Declarative approach. While these two aspects could have been modelled as separate processes (one imperative and the other declarative), using the hybrid approach allows us to join the two and show how they interact. Initially, only Receive Patient is required due to the declarative constraint init. After executing Disinfect Wound, Stitch Wound has to be executed because of a response between them. Registration and treatment can happen in parallel, but prescription of antibiotics is dependent on the patient data.

The model can be extended with time information and exercises to obtain simulation-based performance information. It is also possible to obtain a simulation log from CPN Tools, which can be imported directly into ProM 6.3 for analysis using a known process. CPN Tools also offers state-space analysis for ensuring the absence of errors such as dead-locks in the process. For more information about hybrid modeling, we refer the interested reader to [7].

Domain-specific visualization. While colored Petri net models are graphical, they are also complex to understand for non-experts. Previously, CPN Tools supported visualizations of such models by means of an external tool, but with version 4 such visualizations are internalized, making it possible to show model and visualization side-by-side without requiring external tools. In Fig. 2, we see two simple visualizations of the model from Fig. 1. The sequence diagram (left) shows a single patient interaction and is updated when simulation is conducted. The visualization is driven purely by the model, and as CPN Tools allows users full control over the simulation, can be used to demonstrate complex scenarios

in a simple way. The bar chart (Fig. 2 (right)) shows aggregated statistics over multiple simulations.

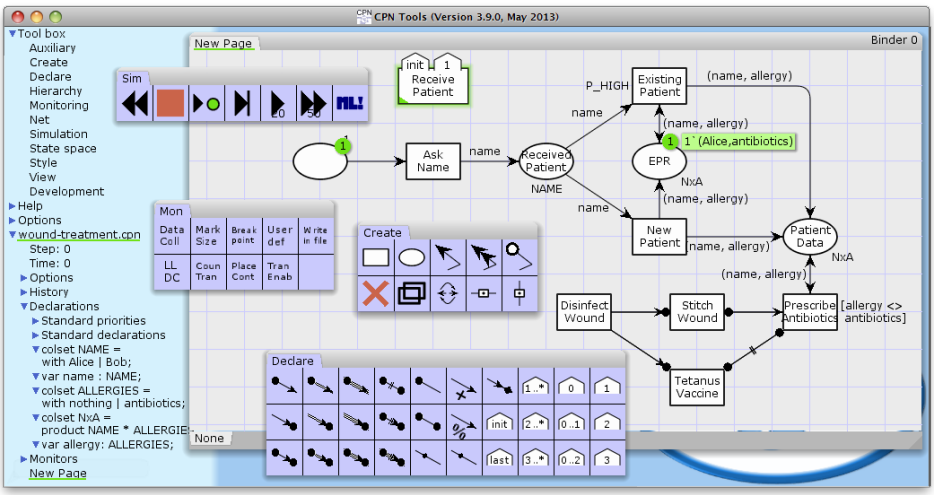

Fig. 1: Overview of CPN Tools with an example hybrid model for a hospital loaded.

Process-partitioned colored Petri nets. Colored Petri nets allow modelers a lot of freedom. Most importantly, it is very hard to separate the flow of data

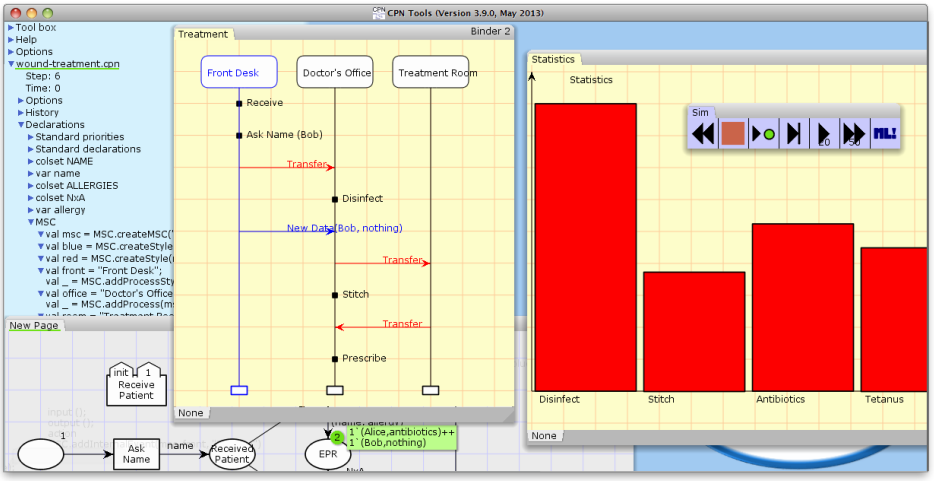

Fig. 2: Two visualizations of the simple model from Fig. 1. The model itself is just visible below the visualizations.

from the flow of control, which makes models hard to understand and analyze. Workflow Nets solved this problem for standard Petri nets, but some of the restrictions are too severe for efficient use of the higher-level formalism. Colored Workflow Nets [1] generalize Workflow Nets to colored Petri nets, but impose some restrictions that make models unnatural. Instead, CPN Tools implements Process-partitioned colored Petri nets (PP-CPNs) [4], which allow more flexibility and more natural models. PP-CPNs explicitly separate the flow of control and data, separating places into process places, local and shared places (for data), resource places, and communication (buffer) places.

PP-CPNs allow multiple instances of multiple process types to communicate, and hence supports an artifact-centric modeling style. Of course, classical Workflow Nets are recognized as PP-CPNs as one would expect. An example PP-CPN model of a simple producer/consumer system can be seen in Fig. 3 (top). Here, we have two kinds of processes communicating over a buffer place;

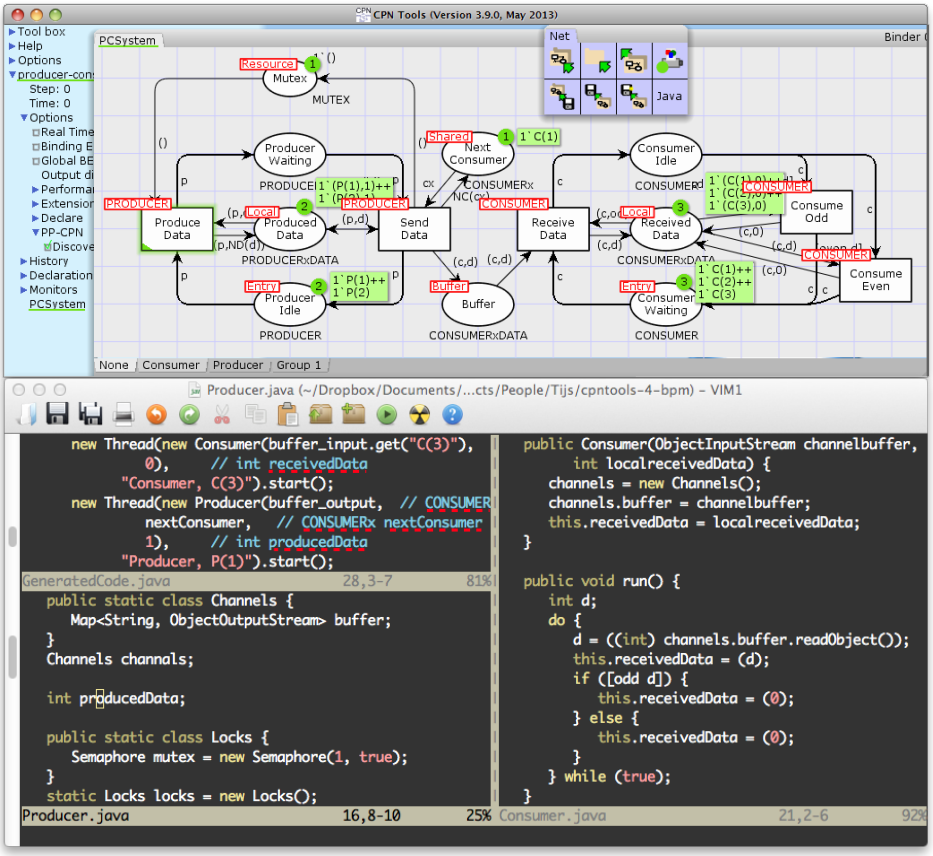

Fig. 3: A colored Petri net model with an explicit process perspective (top) and (some of the) generated Java code from the model (bottom).

producers produce items (integers), store them locally, and transmit them. They use a mutex (a single resource) to prevent race conditions. Initially there are two producers. Consumers receive data from producers, store it locally and dispatch depending on the data.

An advantage of PP-CPNs is that it is possible to generate them automatically from code and to generate running Java code from such models; an example of code generated from the model in Fig. 3 (top) is shown in Fig. 3 (bottom).

Maturity, availability, screencast. CPN Tools is a very mature tool and has been in active use for over 10 years. It enjoyed approximately 5500 downloads in the period May 1, 2012–May 1, 2013. It is used in teaching in several universities, used by companies, and a large number of case studies in several fields are available from `http://cs.au.dk/cpnets/industrial-use/` and on our own homepage we showcase models from industrial case studies at `http://cpntools.org/documentation/examples/`. We are currently conducting case studies using the new declarative constraints, but these are on-going and not yet ready for publication. The implementation of the Declare language is an optimized version of the Declare tool [6].

CPN Tools is open source and available for free for everybody at `http://cpntools.org/`. On this page, we also have a comprehensive getting started guide including screencasts for beginners. In the future, we plan to extend CPN Tools with timed and process-aware versions of Declare.

References

1. van der Aalst, W.M.P., Jørgensen, J.B., Lassen, K.B.: Let's Go All the Way: From Requirements Via Colored Workflow Nets to a BPEL Implementation of a New Bank System. In: Proc. of OTM Conferences (1). LNCS, vol. 3760, pp. 22–39. Springer (2005)
2. Hildebrandt, T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. In: Post-proc.of PLACES 2010 (2010)
3. Hildebrandt, T., Mukkamala, R.R., Slaats, T.: Nested dynamic condition response graphs. In: Proc. of Fundamentals of Software Engineering (FSEN) (April 2011)
4. Kristensen, L.M., Westergaard, M.: Automatic Structure-Based Code Generation from Coloured Petri Nets: A Proof of Concept. In: Proc. of FMICS. pp. 215–230. LNCS, Springer (2010)
5. Westergaard, M.: CPN Tools 4: Multi-formalism and Extensibility. In: Proc. of ATPN. LNCS, vol. 7927, pp. 400–409. Springer (2013)
6. Westergaard, M., Maggi, F.M.: Declare: A Tool Suite for Declarative Workflow Modeling and Enactment. In: Business Process Management Demonstration Track (BPMDemos 2011). CEUR Workshop Proceedings, vol. 820. CEUR-WS.org (2011)
7. Westergaard, M., Slaats, T.: Mixing Paradigms for More Comprehensible Models. In: Proc. of BPM. LNCS, vol. 8094. Springer (2013)

Enacting Complex Data Dependencies from Activity-Centric Business Process Models

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

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

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

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

1 Introduction

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

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

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

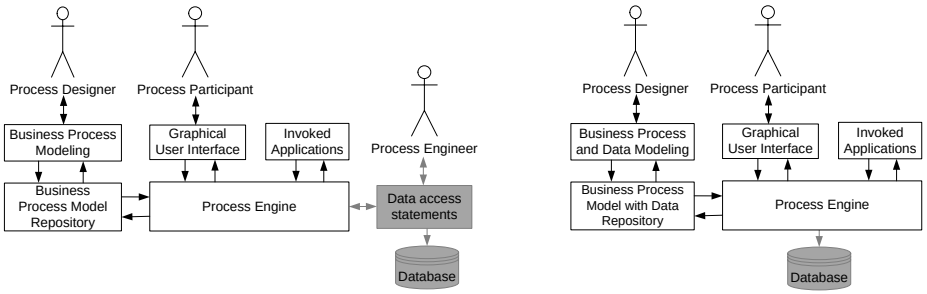

Fig. 1. Process engine architecture: classical (left) and proposed (right).

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

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

2 Modeling Complex Data Dependencies in BPMN

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

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

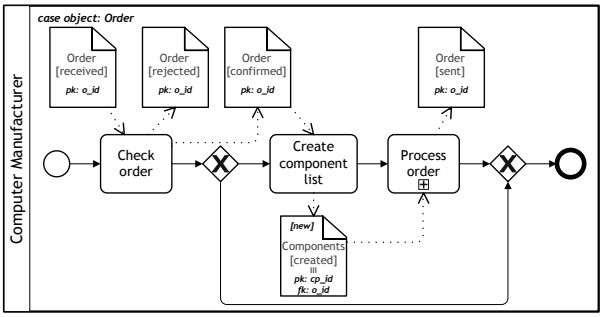

Fig. 2. Build-to-order Process of a Computer Manufacturer

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

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

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

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

3 Tool Architecture and Implementation

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

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

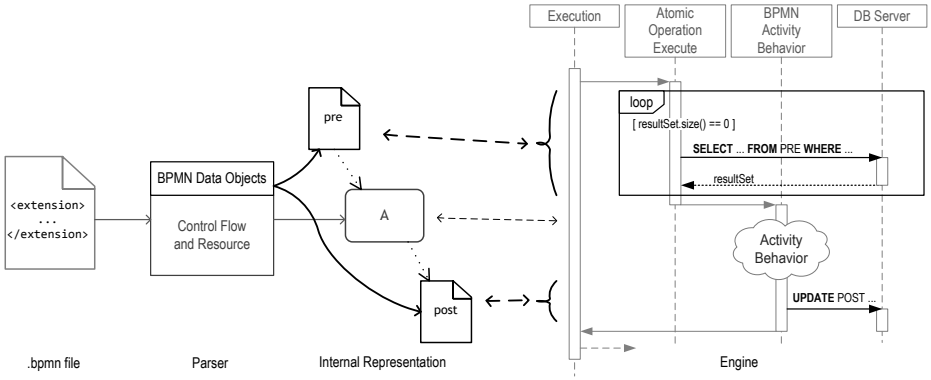

Fig. 3. Data dependencies are a conservative extension to data format, parser, internal representation, and execution engine.

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

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

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

4 Conclusion

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

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

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

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

References

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

An Extensible Platform for Process Model Search and Evaluation

Christian Ress and Matthias Kunze

Hasso Plattner Institute at the University of Potsdam,
christian.ress@student.hpi.uni-potsdam.de,
matthias.kunze@hpi.uni-potsdam.de

Abstract. We present a platform that integrates a number of process model search techniques and provides a uniform interface for query formulation and search result presentation, as well as a framework to evaluate a particular search technique. The platform provides researchers with a common infrastructure to embed their search technique in form of a dedicated search engine. The demo explains the features of our platform and encourages researchers to contribute their search techniques.

1 Introduction & Background

Business process models are a central cornerstone of process-oriented organizations as they explicitly capture the knowledge to carry out the operations of a business and are reused for documentation, analysis, automation, and certification, among others. Modern companies maintain thousands of process models for reference and reuse [2], which requires effective capabilities to search among them. Researchers have proposed an abundance of techniques that focus on text, structure, or behavior of process models and allow for similarity search, i.e., obtaining approximate matches to a complete model, and querying, i.e., searching precisely by few yet relevant aspects of a model [2].

The majority of these process model search techniques, however, has been presented only theoretically. Evaluations of their quality and performance has been conducted under lab conditions, i.e., with a minimalistic implementation that mainly addresses the query processing, i.e., comparing candidate models with the query and deciding a match. This is, arguably, due to the effort of providing a complete process model search infrastructure that includes a user interface to formulate a query, a process model repository to store, manage, and retrieve models from, and the visual presentation of search results as a response to the query. This functionality is shared among all process models search techniques.

To this end, we have developed a prototypical process model search platform that assumes these tasks and allows for the integration of dedicated search techniques in form of a search engine plugin architecture. This includes a set of well-defined APIs that integrate a search engine with our platform. Moreover, the platform provides a framework to evaluate a search engine with regards to the quality of a search technique, i.e., the relevance of the provided results, and, the performance of its implementation. The platform aims to reduce the time to implement and evaluate a particular search technique, and enables the

comparison of various techniques, as they can now be deployed in the same runtime environment.

2 Search with your Search Technique

The central concept of our search platform is to enable developers to deploy a dedicated search engine to the platform and use it to search for process models in a straight-forward manner. Hence, one of the key features of our search platform is a presentation layer, which lets users specify search queries using BPMN and view ranked search results in a similar visual representation, depicted in Fig. 1.

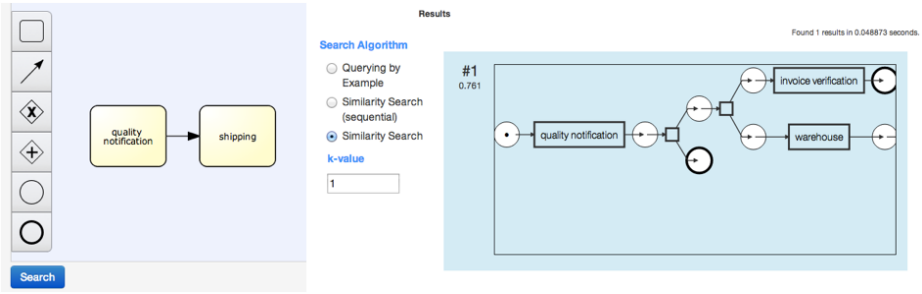

Fig. 1: Screenshot of the web-search interface.

The presentation layer includes a simplistic, web-based process model editor, that allows formulating queries as regular process models, as this is typical for similarity search and has been proposed for querying [6], too. The editor itself is highly extendable, which allows the formulation of queries in languages devised for search, e.g., BPMN-Q [1]. The input query is provided to the dedicated search engine that parses it and matches it against stored process models. To this end, BPMN queries are transformed to Petri nets.

The extensible architecture of the search platform makes it possible to integrate a dedicated search engine, provided it is implemented in Java, by providing common interfaces through which the search platform can communicate with the search engine. Currently, we require that search algorithms accept queries in form of Petri nets, defined using the jBPT Java library[1], and return search results in a similar format. We resorted to Petri nets, as they provide a common formalism for a variety of business process modeling languages [7].

Our aim is to provide researchers with an opportunity to experiment with their algorithms faster and easier, and explore the search results in an environment that is similar to one that end users expect. We do this by providing an API through which search algorithms can expose parameters that can be changed during runtime, and make these parameters accessible in the search interface. This way, parameters can be configured without modifying source code or static configurations, and without recompilation. The results of the chance are visible immediately.

[1] `http://code.google.com/p/jbpt/`

3 Evaluate your Search Technique

Our platform has originally been devised to integrate a number of dedicated search techniques in a common infrastructure. However, it turned out that the very same functionality of a search engine can be used to evaluate the underlying search technique and its implementation. That is, experiments are typically carried out by running a well-defined set of queries against a set of candidates, and assessing quality and performance. Thus, we developed a framework that allows running predefined experiments against search engines without the need for a complex evaluation infrastructure.

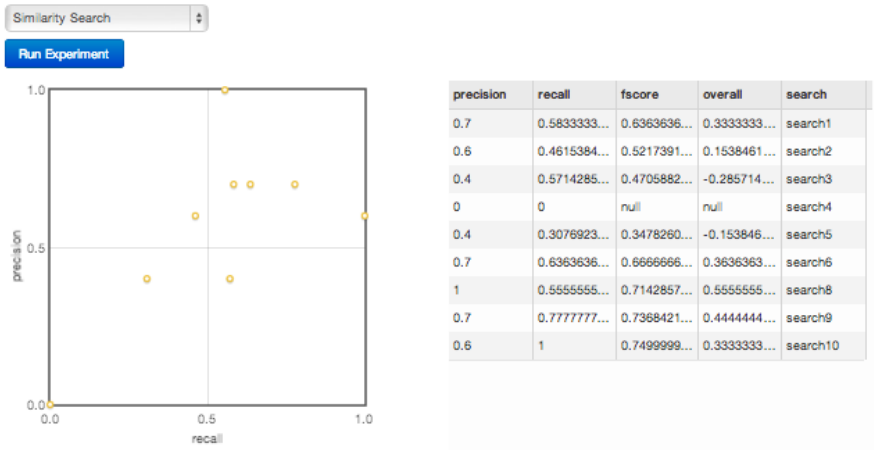

Fig. 2: Screenshot of the interface for precision/recall analysis.

Two methods for evaluating a search technique are provided. *Quality* judges on the relevance of matches in a search result with respect to a human assessment and therefore needs a reference data set. For similarity search, such a dataset has been introduced in [3]. *Performance* addresses the resource consumption of the implementation of a search engine and its scalability with regard to large process model collections. Hence, performance can be evaluated without a human data set, using any process model collection.

For this purpose, a number of time measurements and counters are provided through an API, which can be uses during the execution of a search run by a dedicated search engine. Features, such as support for laps and persistence of counters and timers over multiple search requests are available. All measurements are automatically included in the response of a query, along with statistical measures such as average, median, quantiles, standard deviation, etc.

To evaluate a search technique, we developed a web-based evaluation interface that allows choosing among a set of quality and performance evaluation methods, e.g., computation of precision and recall values and the visualization of precision-recall curves. With regards to performance measures, trend analyses can be

plotted over a number of search runs for various sizes of the candidate model collection.

Fig. 2 shows an excerpt of the evaluation interface that allows choosing a dedicated search engine and compute precision and recall values for each of the queries. The result is provided in a table for each query, and a visualization shows the queries in a coordinate system. This allows for fast identification of queries with significantly good (right upper quadrant) or poor (left lower quadrant) quality.

4 Architecture

The search platform has been implemented as a two-tiered web application, consisting of an HTML5 frontend and a Java backend, depicted in Fig. 3. The web-search and evaluation interfaces are implemented as web applications that run in common web browsers and require no installation. They communicate with the search platform via a JSON API and prove for the interaction with the user. For *search*, a simplistic process model editor based on the processWave editor[2] is provided to formulate the query. Search results are provided as an ordered list along with quality measures, cf. [4]. *Evaluation* offers predefined experiments, i.e., a set of queries and candidates, run against a dedicated search engine and visualizes results in terms of quality and performance. Particular techniques to match a query with models from the process model repository are realized in dedicated search engines.

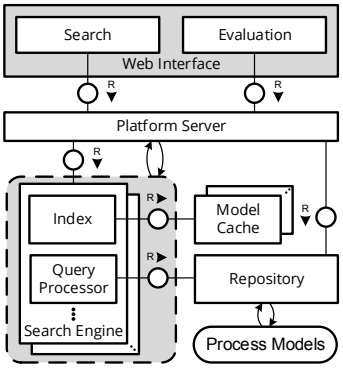

Fig. 3: Architecture of Process Model Search Platform

Search and Evaluation interfaces communicate with the platform server. As an evaluation comprises running reference searches, the platform server does not distinguish between search and experiment. That is, the experiment framework is implemented completely client-side. The search platform server integrates different dedicated *search engines*. Such an engine comprises at least a query processor that decides, whether a candidate matches the query and scores relevance. A custom index enables efficient search. To facilitate the implementation of these components, the search platform provides shared components that can be accessed by the components of a search engine, i.e., a model cache that underpins a custom index and a persistence interface with the repository that manages stored process models. The model cache increases startup speed as it preserves data that has been expensively precomputed, when models are loaded.

Through our strict use of a generic JSON API to access the search platform server it could also be used as a web service for process search and be integrated with other services or applications.

[2] `http://processwave.org`

5 Maturity & Show Case

The search platform has been implemented as a prototype to elaborate on the requirements on search engines and their integration with a common platform. Since dedicated search methods require their own query processor and indexes, such a platform provides users with a unique search interface that covers various perspectives of process model search, including similarity search and querying. At the current state, we have implemented similarity search and querying based on successor relations, cf. [5,6]. As matching is conducted on net-based formalisms, search results are currently presented using their Petri net representations. This shall be extended in future work. Also, the quality experiments are limited to human assessment of similarity, cf. [3], as other reference data was not available.

In the demo, we address researchers that are interested in process model search and may even have proposed a search technique on their own. We will introduce the platform and its architecture in a brief presentation before turning to a live demo that comprises two parts.

1. We demonstrate the search and evaluation capabilities of the platform by means of example queries, and their results. This includes a discussion of search result quality metrics and how they support users in understanding a search result. For evaluation, we show how various measures and diagrams give insight into the quality and performance of a search technique.
2. In a quick walkthrough tutorial, we explain the requirements of a custom search engine and which steps are required to integrate it into the platform by a simple example.

A screencast demonstrating our search and evaluation platform can be found at: `http://vimeo.com/ress/process-search-demo`. The platform is publicly available under the MIT open source license along with a short tutorial on how to use it and integrate a custom search engine at `http://bitbucket.org/ress/process-search`.

References

1. A. Awad. BPMN-Q: A Language to Query Business Processes. In *EMISA*, volume 119, pages 115–128, 2007.
2. R. Dijkman, M.L. Rosa, and H.A. Reijers. Managing Large Collections of Business Process Models—Current Techniques and Challenges. *Comput Ind*, 63(2):91, 2012.
3. R. Dijkman, M. Dumas, B. Dongen, R. Käärik, & J. Mendling. Similarity of Business Process Models: Metrics and Evaluation. *Inform Syst*, 36(2):498 – 516, 2011.
4. M. Guentert, M. Kunze, and M. Weske. Evaluation Measures for Similarity Search Results in Process Model Repositories. *ER '12*, pages 214–227, Springer, 2012.
5. M. Kunze, M. Weidlich, and M. Weske. Behavioral Similarity—A Proper Metric. In *BPM '11*, pages 166–181. Springer, 2011.
6. M. Kunze and M. Weske. Local Behavior Similarity. In *BPMDS '1*, volume 113 of *LNBIP*, pages 107–120. Springer, 2012.
7. N. Lohmann, E. Verbeek, and R. Dijkman. Petri Net Transformations for Business Processes—A Survey. In *Transactions on Petri Nets and Other Models of Concurrency II*, pages 46–63. Springer, 2009.

Worklr: Supporting and Capturing Business Processes from Knowledge Workers

David Martinho and António Rito Silva

[1] Instituto Superior Técnico – UL
[2] ESW - INESC-ID
davidmartinho,rito.silva@ist.utl.pt

Abstract. Worklr (Workflow enhanced with Live Recommendations) is a workflow tool that relies on one of the most known user experience (UX) pattern: the exchange of email messages. In organizations that do not have workflow systems guiding their business processes, knowledge workers rely on technologies like telephone, email or fax, to interact and attain their business process goals. Knowledge workers operate the business in a daily basis, and they know how to handle the most complex business situations. Worklr supports the capture of workers' knowledge following a design-by-doing approach. Workers attain business goals by producing data objects. The convergence of workers' behavior is fostered through a recommender system. This novel proposal leverages on data-driven workflow approaches.

Keywords: Ad-hoc Workflow, Operational Support, Recommender Systems

1 The Worklr Tool Approach

Our approach to capture knowledge workers' tacit knowledge is based on an ad-hoc workflow system which implements conversational acts, and where business goals are achieved when a set of data objects is produced [1]. Knowledge workers are responsible to produce data objects towards the complete generation of the information required for the process goal achievement. Additionally, knowledge workers can establish conversational acts to request the production of data objects by their co-workers. Associated to a request, the initiator may include some input data objects, and the executor is expected to reply with output data objects that she produced to fulfill the request. The flow of conversational acts (i.e. requests), supported by the ad-hoc workflow system, is driven by the knowledge workers' knowledge of their own organizational responsibilities and their co-workers responsibilities: they know who is responsible to produce what.

Consider a Travel Process case where an *employee* wants to go on a business travel to Beijing. First, he creates a process instance named `Travel Process`. As he initiates such process instance, the employee is automatically assigned to execute the first request of the process, a.k.a. root request. It is within this root

request, automatically named after the process name, that the employee will create any necessary sub-requests.

The employee knows that the secretary is responsible to handle his trip, but before he sends her a sub-request, he creates a set of data objects that he knows the secretary will need. Therefore, the employee creates a data object labeled `Destination` and fills its value with Beijing, two data objects labeled `Departure Date` and `Returning Date` with the values 25/08/2013 and 30/08/2013, respectively, and a data object labeled `Motivation` where he states that he wants to attend the BPM2013 conference. For now, these four data objects are defined as the creation context of the request identified by the request label `Travel Process`[3].

Having the necessary data objects, the employee is in condition to send a sub-request to the secretary. To do so, he creates a new sub-request, chooses the secretary's queue as destination, names the request `Travel Request`, adds a little description to communicate the motivation for that request as a commentary, and selects all the data objects he created before: `Destination`, `Departure Date`, `Returning Date` and `Motivation`. As the employee sends the sub-request, that request is published in the secretary's work queue.

As one secretary claims the request for execution, she is proclaiming herself as the executor of that request, being able to see the data objects provided by the request's initiator. Having the information provided by the employee, the secretary calls some travel agencies to request some hotel and flight prices. However, before booking any hotel room or flight, the secretary needs an authorization from the *supervisor*. Similarly to the employee, the secretary also knows that the supervisor needs to know the trip details and the best tender value she got from the agencies. As such, she creates a data object named `Tender` and fills it up with the best price she got from the travel agencies. Afterwards, the secretary creates a new sub-request, which she addresses to the supervisor's queue, labels the request `Travel Authorization`, writes a comment on what she needs from the supervisor, and, as input, she selects the data objects labeled `Motivation`, `Departure Date`, `Returning Date`, `Destination` and `Tender`.

In the same way, as the request is sent, it is published in the supervisor's queue. As one of the supervisors claims this request for execution, he can see the data objects that the secretary attached as input of the request. Based in this data, the supervisor executing the request decides to either authorize or not the travel. To let the secretary know that she can proceed with the hotel and flight booking, the supervisor creates a new data object labeled `Authorization` and defines its value as *Granted*. Then, the supervisor is in condition of responding to the request and let the secretary know of his decision. Notice that the secretary will only see the data object if the supervisor explicitly selects it when he responds to the request. After the supervisor responds to the request, he cannot create any sub-request or data objects in that context[4].

[3] The root request is created automatically and named after the process name

[4] Although this requirement appears to be limitative, its for correctness purposes

After the supervisor replies to the `Travel Authorization` request, the secretary can see the responded data object `Authorization` along with its `Granted` value. Based on that positive value, the secretary calls the travel agency and books the hotel and flight, obtaining the respective reservation number and flight ticket number. With this information, she creates two data objects labeled `Hotel Reservation` and `Flight Number` and fills in the respective information. After this, she is in condition to respond to the original request initiated by the employee, providing both the `Hotel Reservation` and `Flight Number` data objects.

As the secretary responds to the request, the employee can see the hotel and flight information contained in the data objects included in the response. Since he needs no more information, and has no pending or executing sub-requests, the employee completes the process.

With the process completed, the employee is essentially saying that the process attained its goal, which consists in getting the travel approved and both the hotel and flight information. During the process execution, several data objects were produced and drove the process instance. Worklr considers all the created data objects and stores their set of labels, along with their cardinality, as a business process goal. This means that completing the process depicted in the example above results in a process goal with the following eight data object labels:

- 1 x `Destination`
- 1 x `Motivation`
- 1 x `Departure Date`
- 1 x `Returning Date`
- 1 x `Tender`
- 1 x `Authorization`
- 1 x `Hotel Reservation`
- 1 x `Flight Number`

Having this process goal is important to guide future business process instances under the same label, i.e. process instances labeled `Travel Process`. Worklr knows, that if in the past a particular process goal was attained, it is likely that same goal will be attained again in a future execution of that kind of process. Hence, along with other contextual information gathered during the execution of a business process instance, Worklr uses such business process goal information and inspects the current state of the process to recommend the creation of new data objects or sub-requests. Such recommendations are based on a process goal and the current execution context, i.e. the user's roles, the labels of data objects available to the user, and the overall process' data object labels.

Therefore, apart from providing operational support, Worlr is also capable of storing previous executions in a structured way, which contain information to guide future process instances with the same label. Nevertheless, as new business situations occur (e.g. if the supervisor refuses the authorization), the Worklr ah-hoc aspect supports that change, i.e. it does not enforce recommendations,

and stores the new attained process goal. That new process goal is taken into consideration in the following executions of processes labeled `Travel Process`.

1.1 Features

The tool is in its first prototype stage, and its source code is not available yet due to intellectual property restrictions inherent to the PhD status of one of the authors.

The following list highlights a set of core functionalities of the Worklr application, regarding operational support:

- Create new business process instances.
- Create data objects within the execution of a request.
- Edit data objects created within the execution of a request.
- Create sub-requests and publish them into user and role queues.
- Claim requests received in the Inbox for execution.
- Communicate with the initiator of the executing request.
- Communicate with the executor of the sub-request.
- Cancel pending and executing sub-requests.
- List all sub-requests initiated and see their respective status (pending, executing, completed, canceled).
- Provide data objects as input of sub-requests.
- Respond to executing requests, specifying which data objects should be included in the response.
- Complete business process instances.

Apart from operational support, Worklr saves executions in the form of *request templates*, which are entities that abstract the different completed requests by storing some contextual information. A request template is therefore composed by:

- an *initiator role context*, which is essentially the set of organizational roles played by the initiators of that kind of request.
- an *input data context*, which is the set of data object labels, and respective cardinality, of the data objects provided as input in that kind of request.
- a *creation data context*, which is also a set of data object labels, and respective cardinality, created in that kind of request.
- an *output data context*, which, analogously, is the set of data object labels, and respective cardinality, responded in that kind of request.

Along with all the business process goals attained under a particular process label, e.g. `Travel Process`, the Worklr tool provides a recommender system that analyses the context of execution of the current user, and computes a set of recommendations of both the labels of the data objects that should be created, and the labels of the sub-requests that can also be created. To each recommendation is associated a score, as discussed in [2], from which the list of recommendations is ordered accordingly.

Hence, the list of Worklr features is extended by this recommender system, and the tool can also:

– Provide sub-request recommendations based on the current execution context.
– Provide data creation recommendations based on the current execution context.

As users perceive recommendations as useful, they are re-using the labels and behavior from other organizational parties playing a similar set of roles. Additionally, although the number of process, request and data object instances increase as more cases are handled, if recommendations are followed, the number of request templates and data object templates converges and recommendations become more accurate.

1.2 Experiment

As shown in the screencast of the tool[5], Worklr is in a functional state enough to conduct an experiment. In [3], we discuss the results of an experiment that we recently conducted, where we gathered some important feedback.

In the experiment, we had 13 participants and 1 confederate. The confederate would initiate new process instances and the root request, while the 13 participants were distributed across 4 distinct organizational roles that should cooperate to attain a particular business process goal. The business process used in the experiment was similar to the one exemplified here, but with more business rules.

A last comment on the experiment: we gathered some important results from the tool as we perceive some convergence in the use of labels by different participants. In the cold-start of the experiment, users would use different labels to identify requests and data objects, but after 16 business process instances, we identified some interesting convergence results.

Acknowledgement

This work was supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, under project PEst-OE/EEI/LA0021/2013.

References

1. Martinho, D., Silva, A.: Non-intrusive capture of business processes using social software. In: BPM Workshops, pp. 207–218. Springer (2012)
2. Martinho, D., Silva, A.: A recommendation algorithm to capture end-users' tacit knowledge. Business Process Management 7481, 216–222 (2012)
3. Martinho, D., Silva, A.R.: An experiment on capturing business process from knowledge workers, submitted to the BPMS2 Workshop at BPM2013

[5] http://vimeo.com/user4862900/worklr-demo-bpm2013 - Password: BPM2013

Declarative Process Mining with the Declare Component of ProM

Fabrizio Maria Maggi*

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

Abstract. Traditional process mining techniques mainly work with procedural modeling languages (like Petri nets) where all possible orderings of events must be specified explicitly. However, using procedural process models for describing processes working in turbulent environments and characterized by a lot of variability (like healthcare processes) is extremely difficult. Indeed, these processes involve several possible execution paths and representing all of them explicitly makes the process models quickly unreadable. Using declarative process models (like Declare) ensures flexibility in the process description. Even processes that work in environments where participants have more autonomy and are, therefore, more unpredictable can be represented as compact sets of rules. In the process mining tool ProM, there are several plug-ins that can be used for different types of analysis based on Declare ranging from the discovery of Declare models, to conformance checking, to the online monitoring of running process instances.

1 Introduction

Process discovery, conformance checking and process model enhancement are the three basic forms of process mining [9]. In particular, process discovery aims at producing a process model based on example executions in an event log and without using any a-priori information. Conformance checking pertains to the analysis of the process behavior as recorded in an event log with respect to an existing reference model. Process model enhancement aims at enriching and extending existing process models with information retrieved from logs, e.g., a process model can be extended with performance-related information such as flow time and waiting time.

In the last years, several works have been focused on process mining techniques based on declarative process models [2–4, 6–8]. These techniques are very suitable for processes characterized by high complexity and variability due to the turbulence and the changeability of their execution environments. The dichotomy procedural versus declarative can be seen as a guideline when choosing

* The author would like to thank R.P. Jagadeesh Chandra Bose, Andrea Burattin, Massimiliano de Leoni and Luciano García-Bañuelos for their collaboration in the implementation of the plug-ins presented in this paper.

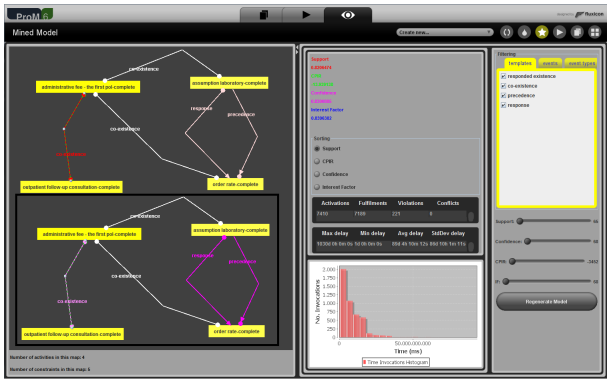

Fig. 1. Screenshot from the Declare Maps Miner.

the most suitable language to represent models in process mining algorithms: process mining techniques based on procedural languages can be used for predictable processes working in stable environments, whereas techniques based on declarative languages can be used for unpredictable, variable processes working in turbulent environments.

The plug-ins presented in this paper use the *Declare* notation [10]. Declare is a declarative language that combines a formal semantics grounded in Linear Temporal Logic (LTL) on finite traces. A Declare model consists of a set of constraints which, in turn, are based on templates. Templates are abstract entities that define parameterized classes of properties and constraints are their concrete instantiations.

In this paper, we illustrate how the presented plug-ins work by showing their application in the context of the BPI Challenge 2011 [1] that records the treatment of patients diagnosed with cancer from a large Dutch hospital. This demonstrates that they are viable to handle event logs of real-life size.

2 Plug-ins Description

Through the *Declare Maps Miner*, it is possible to generate from scratch a set of Declare constraints representing the actual behavior of a process as recorded in an event log. The user selects from a list of Declare templates the ones to be used for the discovery task, i.e., the mined model will contain only constraints that are instantiations of the selected templates. In addition, the user can provide the plug-in with some a-priori knowledge (if it is available), to guide the discovery task. For example, the user can decide to limit the possible constraints to discover. This allows her to generate only constraints that are the most interesting from the domain point of view, thus reducing the complexity of the resulting models.

There are two mechanisms for limiting the possible constraints to discover. First, it is possible to ignore constraints between event types (such as *start* and

complete) of the same activity.[1] Secondly, the user can provide different groups of activities (each group can be loaded in the form of a list of activities in a text file) and specify if only intra-group or inter-group constraints should be considered. Intra-group constraints refer to the class of constraints where the activities involved all emanate from a single group. In many scenarios, analysts would be interested in finding constraints between activities pertaining to a functionality, to a particular department in an organization, etc. For example, in a hospital event log, an analyst would be interested in finding relationships/constraints between the various administration activities. Inter-group constraints refer to the class of constraints where the activities involved belong to two different groups. For example, in a hospital log, an analyst would be interested in constraints between activities involved in surgery and therapy.

The user can also specify thresholds for parameters *minimum support* and *alpha*. Parameter minimum support allows her to select the percentage of traces in which a constraint must be satisfied to be discovered (and to filter out noisy traces). Parameter alpha can be used ignore constraints that are trivially true in the discovery task. For example, constraint *response(A,B)*, specifying that if *A occurs*, then eventually *B follows*, is trivially true in process instances in which *A* does not occur at all.

Fig. 1[2] shows the output produced by the Declare Maps Miner. The discovery results are presented to the user as interactive maps and give different information about the process. Activities (shown as rectangles) are colored based on their frequency (from white indicating low frequency to yellow indicating high frequency). The user can highlight, in the discovered maps, the Declare constraints (shown as arcs between activities) that are more relevant (based on different metrics) or prune out the ones that are less interesting, redundant or deriving from noise in the log. The maps produced by the Declare Maps Miner also show delays, latencies, time distances between activities and several statistics related to the temporal dimension of the process.

If an existing Declare model is already available, it is possible to repair it based on the information retrieved from a log (plug-in: *Repair a Declare Model*). The user can decide which aspects must be kept in the original set of rules and which ones can be discarded. For example, the user can decide to keep only certain types of constraints or constraints involving certain activities. It is possible to just remove less interesting or redundant rules or to strengthen (if possible) the constraints included in the original set. Starting from a Declare model and from a log, the Declare model can be extended with time information (plug-in: *Extend a Declare Model with Time Information*). Through the *Data-Aware Declare Miner*, it is also possible to discover constraints enriched with data conditions.

[1] The miner is able to deal with non-atomic activities and to discover constraints involving different parts of the activities' lifecycle.

[2] For space reasons, the quality of the screenshots here presented is not the best. However, all the figures included in this paper are available at `http://math.ut.ee/~fabrizio/BPMdemo13/demo/FiguresDemo2013.zip`

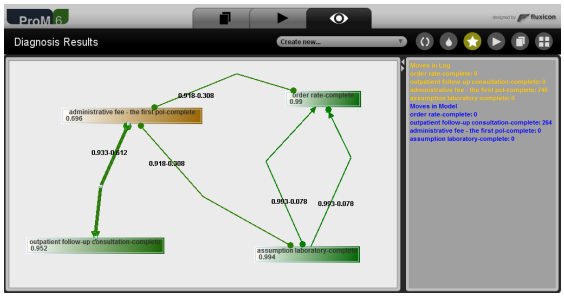

Fig. 2. Screenshot from the Declare Diagnoser.

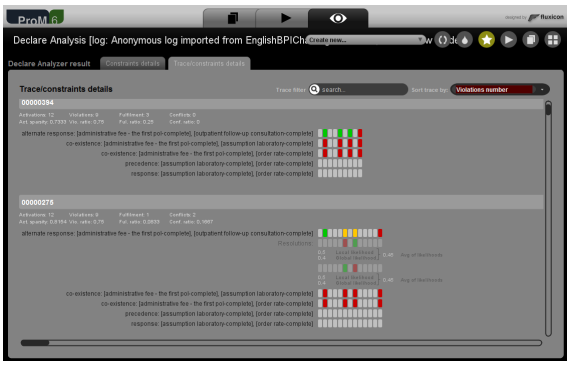

Fig. 3. Screenshot from the Declare Analyzer.

The *Declare Replayer* and the *Declare Diagnoser* (*Declare Checker* package) can be used to check the conformance of the actual behavior of a process as recorded in an event log with respect to a reference Declare model. The Declare Replayer generates a set of alignments between the log and the Declare model, i.e., information about what must be changed in the log traces to make them perfectly compliant with the model. The Declare Diagnoser projects the results obtained through the Declare Replayer onto the reference model (as shown in Fig. 2). This projection produces a map in which the critical activities/constraints of the reference model are highlighted. In particular, activities are colored from red to green according to the number of moves they are involved in, i.e., according to how many times they were in the wrong place in the log or missing when required. Constraints are colored from red to green based on the number of moves that are needed to make the log compliant to them. Through the *Declare Analyzer* (Fig. 3), the user can pinpoint where the process execution deviates from the reference Declare model and quantify the degree of conformance of the process behavior through several metrics, e.g., *fulfillment ratio* and *violation ratio*.

For completeness, we mention that, with *Mobucon LTL* (a provider of the operational support service in ProM), it is also possible to monitor a running

process with respect to a reference Declare model and get information about its compliance at runtime. However, this functionality has already been presented in [11].

3 Summary

The plug-ins belonging to the Declare component of ProM are advanced prototypes and, as demonstrated through their application to the log provided for the BPI challenge 2011, are able to handle logs of real-life size. The Declare Maps Miner and Mobucon LTL have been used in the Poseidon project (`http://www.esi.nl/short/poseidon/`) to monitor sea vessels. In particular, in [5], we show how it is possible to construct Declare models from real-life historical data and monitor live data with respect to the mined model.

We consider the set of ProM plug-ins presented here stable enough for evaluating the declarative approach in real-life case studies. Binaries and source code can be obtained from the ProM repository (`https://svn.win.tue.nl/repos/prom`) and a release of each plug-in including the functionalities described in this paper is included in the nightly build version of ProM (`www.processmining.org`). A screencast of this demo can be found at `http://math.ut.ee/~fabrizio/BPMdemo13/demo/demo.html`. A flyer with a summary of the functionalities described in this paper is available at `http://math.ut.ee/~fabrizio/BPMdemo13/demo/DeclareFlyer.pdf`.

References

1. 3TU Data Center. BPI Challenge 2011 Event Log, 2011. doi:10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54.
2. A. Burattin, F.M. Maggi, W.M.P. van der Aalst, and A. Sperduti. Techniques for a Posteriori Analysis of Declarative Processes. In *EDOC 2012*.
3. Massimiliano Leoni, Fabrizio Maria Maggi, and WilM.P. Aalst. Aligning event logs and declarative process models for conformance checking. In *BPM 2012*.
4. F. M. Maggi, M. Westergaard, M. Montali, and W. M. P. van der Aalst. Runtime verification of LTL-based declarative process models. In *RV 2011*.
5. Fabrizio M. Maggi, Arjan J. Mooij, and Wil M. P. van der Aalst. *Analyzing Vessel Behavior using Process Mining*, chapter Poseidon book.
6. F.M. Maggi, J.C. Bose, and W.M.P. van der Aalst. Efficient discovery of understandable declarative models from event logs. In *CAiSE 2012*.
7. F.M. Maggi, R.P.J.C. Bose, and W.M.P. van der Aalst. A knowledge-based integrated approach for discovering and repairing declare maps. In *CAiSE 2013*.
8. F.M. Maggi, M. Montali, M. Westergaard, and W.M.P. van der Aalst. Monitoring business constraints with linear temporal logic: An approach based on colored automata. In *BPM 2011*.
9. W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.
10. W.M.P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative Workflows: Balancing Between Flexibility and Support. *Computer Science - R&D*.
11. Michael Westergaard and Fabrizio Maria Maggi. Declare: A tool suite for declarative workflow modeling and enactment. In *BPM (Demos)*, 2011.

Leveraging Super-Scalarity and Parallelism to Provide Fast Declare Mining without Restrictions

Michael Westergaard[1,2⋆] and Christian Stahl[1]

[1] Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands
[2] National Research University Higher School of Economics, Moscow, 101000, Russia
m.westergaard@tue.nl, c.stahl@tue.nl

Abstract. UnconstrainedMiner is a tool for fast and accurate mining Declare constraints from models without imposing any assumptions about the model. Declare models impose constraints instead of explicitly stating event orders. Constraints can impose various choices and ordering of events; constraints typically have understandable names, but for details, refer to [5]. Current state-of-the-art mining tends to fail due to a computational explosion, and employ filtering to reduce this. Our tool is not intended to provide user-readable models, but instead to provide all constraints satisfied by a model. This allows post-processing to weed out uninteresting constraints, potentially obtaining better resulting models than making filtering beforehand out of necessity. Any post-processing (and complexity-reducing filtering) possible with existing miners is also possible with the UnconstrainedMiner; our miner just allows more intelligent post-processing due to having more information available, such as interactive filtering of models. In our demonstration, we show how the new miner can handle large event logs in short time, and how the resulting output can be imported into Excel for further processing. Our intended audience is researchers interested in Declare mining and users interested in abstract characterization of relationships between events. We explicitly do not target end-users who wish to see a Declare model for a particular log (but we are happy to demonstrate the miner on other concrete data).

Processes that are not perfectly understood or have little structure, are often easier modeled using a declarative rather than a classical imperative approach. In a declarative approach, constraints between tasks are described rather than for each task specifying the next task to execute. Declarative modeling is a more recent and less matured approach, which has so far not found widespread application in industry yet. Declare [7] is an emerging language studied in academia over the last decade.

Existing miners. Typically the complexity of checking a constraint is $O\left(m \cdot \binom{n}{k} \cdot k!\right)$, where m is the number of traces in the log, n is the number of different events in the entire log, and k is the number of parameters to the constraint (number of ways to assign n events to k ordered parameters – ${}_nP_k = \binom{n}{k} \cdot k!$ – times the number of traces k).

⋆ Support from the Basic Research Program of the National Research University Higher School of Economics is gratefully acknowledged.

For the BPI challenge log from 2012[3] [1] the parameters are $m = 13,087$ and $n = 24$, yielding from $314,088$ ($k = 1$) to $66,749,981,760$ ($k = 5$) checks. ProM 6 has a Declare miner which systematically checks each constraint and returns a model comprising constraints satisfied for a certain percentage of traces. Due to the complexity, this miner employs a couple of tricks, including forcing users to pick among interesting constraints [3], employing a priori reduction to avoid considering rarely occurring events [3], and considering the relationships between constraints to avoid mining some constraints [4]. Even so, the ProM Declare miner did not mine a single constraint such as succession in 24 hours.

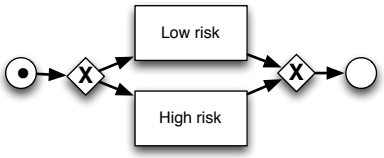

Fig. 1: Model with a branch.

The assumption that rarely occurring events need not be checked is crucially dependent on the notion of *support*, i.e., that a constraint is only interesting if it is triggered. This is problematic in a case such as Fig. 1. There we have a hypothetical XOR-split and -join. If the top path is chosen, constraints in the bottom part are never triggered, so their support is low, even though those are arguably more interesting. If we want to mine without enforcing high support, we cannot use a priori reduction. Exploitation of relationships between constraints is problematic if we want to display a simpler model, e.g., by removing constraints redundant due to transitivity. The problem here is that, generally, support of a removed constraint cannot be derived from support of the remaining constraints.

The MINERful miner [2] uses regular expressions and global computations to mine constraints. First statistics are computed for the log and subsequently constraints are mined from these. The MINERful miner can mine all constraints for the 2012 BPI challenge log in approximately 26 seconds, but only supports a subset of all constraints. Computation of constraints from statistics makes it difficult to add new constraints, as it is necessary to develop and prove rules for doing so. For example, it is far from obvious how to extend this approach to also mine choices.

Removing all assumptions. Instead, we prefer to mine first and, subsequently, filter with full information. Not only does this avoid the problem of support and branches, it allows us full knowledge of all constraints, so we can remove redundant constraints more intelligently using simple patterns. Finally, this allows us to provide users with a slider and let them interactively (and locally) balance between simplicity and support.

Efficient preprocessing. We first make sure that our base mining algorithm is efficient. We transform the log into a simple array of arrays of identifiers instead of actual events. Declare constraints have a (finite) LTL semantics, which can be represented as a finite automaton. Using a precomputed mapping between event identifiers and automaton labels, we obtain a fast and memory-efficient replay of the log. We can mine most of Declare's 34 constraints (excluding 2 with four or more parameters) on the BPI challenge log in 249 seconds using 12 MB of memory (including storing the log and all

[3] We use the 2012 log as the 2013 log is much simpler with $819 \leq m \leq 7554$ and $3 \leq n \leq 4$.

results). We mine the same set of constraints as MINERful in 32 seconds (vs 26). This is done automatically, making it easy to add constraints by entering their LTL semantics.

Symmetry reduction. We observe that some Declare constraints are symmetric in their parameters; this is, for example, the case for the choice 1 of 2 constraint. By only checking these constraints for one ordering of parameters, we halve the number of checks (and time needed). In general, this reduces the checking complexity to $O\left(m \cdot \binom{n}{k} \cdot k! \div \prod_{sg \in SG} |sg|!\right)$, i.e., divided by the faculty of the size of each symmetry group $sg \in SG$. The number we divide by tends to get larger exactly when the traditional approach has problems (i.e., when k is large). For example, the most complex standard Declare constraint is choice 1 of 5, which yields the aforementioned 67 billion checks for the BPI challenge log. As this constraint is symmetric in all parameters, this is reduced to $556,249,848$ checks. Our implementation can mine *all* 34 Declare constraints for the BPI challenge log in 287 seconds (so the reduction allows us to check all constraints in the same amount of time we could only check the easy ones before). We now tie with MINERful at 26 seconds (the reason for less reduction is that MINERful does not handle any constraints where the reduction is large). For the standard constraints, we never have more than two symmetry groups, which means that as number of parameters go up, so does the reduction. We compute the symmetry reduction automatically, making this compatible with any added constraint.

Parallelization. As our miner is simple, we do not need any global computations so far. This means our miner is easy to parallelize. We can parallelize in the number of constraints or the number of traces. With 34 constraints, we can mine each individually. Declare constraints are not equal however, and indeed the choice 1 of 5 constraint is responsible for more than 50% of the total time spent on the BPI challenge log, making this strategy less compelling. As we generate the parameters for the mining deterministically, we can have each computation thread take care of this on its own, alleviating this (but making mining more complex). A simpler approach, and the one we have chosen to go with, is to split the log and have each computation thread handle part of the log. As we have abolished a priori reduction, we do not need any central coordination in this case, and our overhead is so small (< 1 second) we can just run the full mining process for each part of the log. Our results can be directly added, making mining scalable nearly infinitely (with constant overhead for preprocessing and aggregation). This has the added benefit of allowing each thread to use less memory (it only needs to store its own part of the log). Employing this allows us to mine all constraints of the BPI challenge log in 174 seconds (using two cores; the reason scalability is not completely linear is due to the feature Turbo Boost on modern CPUs allowing one core on a multi-core system to run at extra speed if other cores are inactive). We beat MINERful at 19 seconds (vs 26). We have implemented this in a parallel setting (i.e., multiple CPU cores on a single machine), but it is equally applicable in a distributed setting as synchronization is only necessary for aggregating results at the end.

Super-scalar mining. Using the automaton representation from [6], it is possible to check more than one constraint at a time. This comes at a cost of memory as the combined automaton is larger than the individual automata, sometimes even exponentially

so, though in practise the size is always less than 1,000 states. We call this super-scalar mining, using the term from CPUs where it refers to the ability to execute multiple instructions simultaneously. We do not wish to break our symmetry reduction from before. Thus, we build a directed acyclic graph with constraints for nodes and arcs from one constraint to another if the first implies the second holds. We then group by taking each leaf and iteratively grouping it with all predecessors and successors not extending the set of parameters and not having incompatible symmetry groups (e.g., splitting them up so $[[A], [B]]$ is not compatible with $[[A, B]]$—it would split up the symmetry group—but the other way around does hold—the symmetry group is already split). The rationale is that adding new parameters increases complexity as does splitting symmetry groups. Adding a constraint with larger symmetry groups does not increase complexity, while the constraint with larger symmetry groups could be checked more efficient on its own, adding to the super-scalar group comes at no extra cost. We check all such maximal groups. We compute this automatically and tailor it to any constraints selected and any new constraints added. Using super-scalar mining with parallelism, we can mine the entire BPI log in only 57 seconds and even without parallelism, we can mine the entire log in just 92 seconds. We beat MINERful on their constraints at just over 4 seconds, an improvement of a factor 6 over 26 seconds. On a server with 8 slower processing cores, MINERful still runs in 26 seconds, whereas UnconstrainedMiner runs in 30 seconds on the entire set of constraints. Using hyper-threading, the UnconstrainedMiner and MINERful tie in running time with the UnconstrainedMiner mining more and more complex constraints. The gain obtained by super-scalar mining is independent of the log.

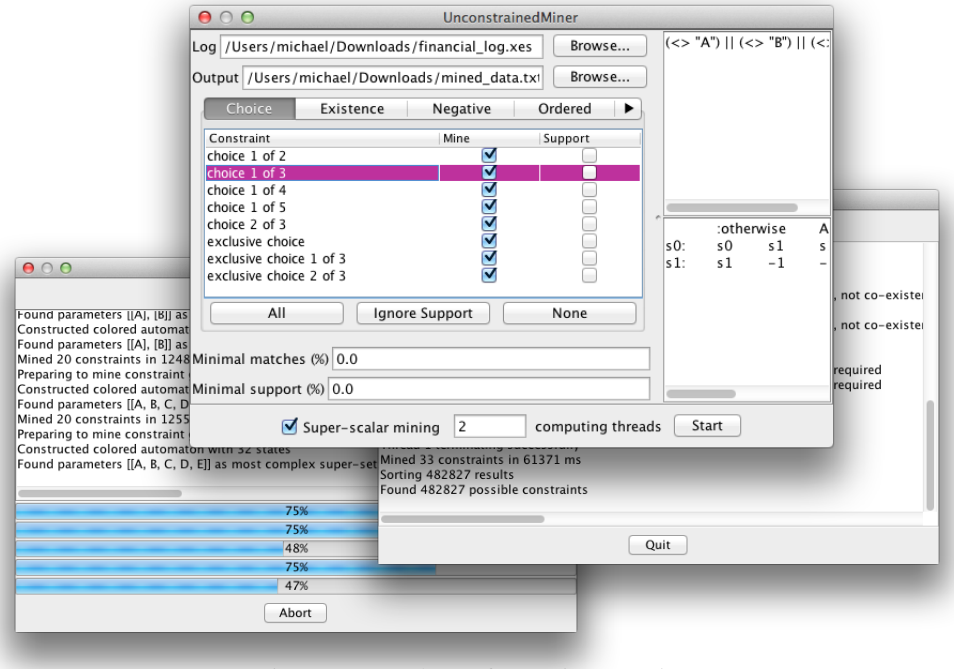

Fig. 2: Screen shots of our miner running.

Usage. In Fig. 2 (top) we see a screen-shot from the configuration of the mining process. All options are saved between runs, making mining from scratch a breeze; the number of threads is computed automatically as the number of cores in the local computer. We can manually inspect the LTL semantics and constraint automaton at the right. In Fig.2 (bottom left) we see the tool in the process of mining. The top progress shows overall progress, and the next progress bars show overall and individual constraint progress for each computation thread. The text area is continuously updated with information about what each thread is doing. At Fig. 2 (bottom right), we see the end result along with total processing time. The end result is a list of all mined constraints, which can be imported in Excel for further processing, as shown in Fig. 3.

Maturity, availability, screencast. Our miner is very new but so far very robust, and uses the same underlying library as the Declare tool (for representing automata) and ProM (for reading logs). The tool is written in Java and runs on all platforms supported by Java. We are currently employing the tool to construct hybrid models containing both declarative constraints and imperative modeling in a block-structured manner.

Our miner is available from `tinyurl.com/declareminer` along with source code. That page also contains a screen cast of simple use of the tool.

References

1. DOI: 10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
2. Di Ciccio, C., Mecella, M.: A two-step fast algorithm for the automated discovery of declarative workflows. In: CIDM 2013. IEEE (2013)
3. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: CAiSE 2012. LNCS, vol. 7328, pp. 270–285. Springer (2012)
4. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: A knowledge-based integrated approach for discovering and repairing declare maps. In: CAiSE 2013. LNCS, Springer (2013)
5. Pesic, M.: Constraint-Based Workflow Management Systems: Shifting Controls to Users. Ph.D. thesis, Beta Research School for Operations Management and Logistics, Eindhoven (2008)
6. Westergaard, M.: Better Algorithms for Analyzing and Enacting Declarative Workflow Languages Using LTL. In: Proc. of BPM. LNCS, vol. 6896, pp. 83–98. Springer (2011)
7. Westergaard, M., Maggi, F.M.: Declare: A Tool Suite for Declarative Workflow Modeling and Enactment. In: Business Process Management Demonstration Track (BPMDemos 2011). CEUR Workshop Proceedings, vol. 820. CEUR-WS.org (2011)

	A	C	D	E	F	G	H	I	J	K	L	M
1	constraint	matches	support	left initial	entered una	returned to i	is initial at er	parameter1	parameter2	parameter3	parameter4	parameter5
2	response	13087	13087	13087	13087	0	13087	A_SUBMITTE	A_PARTLYSUBMITTED			
3	alternate suc	13087	13087	13087	13087	0	13087	A_SUBMITTE	A_PARTLYSUBMITTED			
4	chain succes	13087	13087	13087	0	13087	13087	A_SUBMITTE	A_PARTLYSUBMITTED			
5	alternate pre	13087	13087	13087	13087	0	0	A_SUBMITTE	A_PARTLYSUBMITTED			
6	alternate pre	13087	7635	13087	7635	5452	0	A_SUBMITTE	A_DECLINED			
7	alternate pre	13087	7635	13087	7635	5452	0	A_PARTLYSU	A_DECLINED			
8	alternate pre	13087	7367	13087	7367	5720	0	A_SUBMITTE	A_PREACCEPTED			

Fig. 3: The mining results imported into Excel.

wOIS-paan – Discovering Performer-Activity Affiliation Networking Knowledge from XPDL-based Workflow Models

Hyun Ahn[1], Minjae Park[2], and Kwanghoon Pio Kim[1]

[1] Collaboration Technology Research Laboratory
Department of Computer Science
KYONGGI UNIVERSITY
e-Mail: {hahn, kwang}@kgu.ac.kr
http://ctrl.kyonggi.ac.kr

[2] BISTel, Inc.
e-Mail: mjpark@bistel-inc.com
http://www.bistel.co.kr

Abstract. In this demo-paper, we implement a workflow-supported organizational intelligence system, which is named as wOIS-paan. The major functionality of the current version of the system is to explore "*workflow performer-activity affiliation networking knowledge*" from an XPDL-based workflow model, and to visualize the knowledge in a graphical form of the force-directed-layout of the Prefuse toolkit. The implemented system operates under a series of algorithms discovering, analyzing, measuring, and visualizing workflow performer-activity affiliation networking knowledge from an XPDL-based workflow package[3], which represents involvement and participation relationships, after all, between a group of performers and a group of activities. The eventual goal of the system is to measure and visualize the human resource allotments and contributions in enacting a workflow procedure (or a group of workflow procedures) at a glance. Also, in terms of the scalability of the system, it can be extensible to show the organization-wide workflow procedures. Conclusively, the wOIS-paan system ought to be a very valuable tool for the BPM and workflow design and operational performance analyzers and consultants.

Keywords: workflow-supported social networking knowledge, workflow affiliation networking knowledge, organizational knowledge discovery, workflow intelligence

1 Maturity

In general, a workflow management system consists of two components, the modeling component and the enacting component. The modeling component allows

[3] A group of workflow models is defined as a workflow package in the WfMC's standardization terminology.

a modeler to define, analyze and maintain workflow models by using all of the workflow entities that are necessary to describe work procedures, and the enacting component supports users to play essential roles of invoking, executing and monitoring instances of the workflow model defined by the modeling component. Especially, from the organizational intelligence point of view, the modeling component deals with the *planned* (or workflow build-time aspect) knowledge of organizational resources allocations for workflow-supported operations, while on the other the enacting component concerns about the *executed* (or workflow run-time aspect) knowledge of organizational resources allotments for the workflow-supported operations. With being connected to these view-points, there might be two issues, such as discovery issue[3] and rediscovery issues[4], in terms of the organizational knowledge discovery activities. In other words, the workflow knowledge discovery issue has something to do with exploring the planned knowledge from workflow models defined by the modeling component, and the workflow knowledge rediscovery issue is to explore the executed knowledge from the execution logs of the workflow models. Conclusively, the demo-system, wOIS-paan, is able to discover, analyze, and visualize the *planned knowledge* of workflow performer-activity affiliations and allotments on a workflow model or a group of workflow models.

Availability of the System. The system's development environments are listed as followings. Particularly, we suppose that the XPDL workflow package's release version is XPDL 1.0. So, it is necessary to be refurbished to support the recently released version of XPDL 2.0 or more, which reflects the BPMN[4] graphical constructs.

- Operating System: Windows 7 Ultimate 64bit
- Programming Language: Java Development Toolkit v.6.0
- XPDL Version: XPDL 1.0
- Development Tool: Eclipse Indigo Release 2
- Libraries: Awt/Swing, Prefuse, Xpdl

However, the system's execution environments are any types of operating systems, and the executable system is available on the website of the authors' research group, the collaboration technology research lab, at the department of computer science, Kyonggi University, https://ctrl.kyonggi.ac.kr/wois.html, and anyone can download the executable system and its demo workflow models in XPDL after registering as a member of the wOIS-paan's user group.

Use Cases and Features. The workflow performer-activity affiliation networking knowledge can be not only discovered from a workflow model defined by the modeling component, but also rediscovered from its execution event logs stored by the enacting component. In this demo-paper, we focus on the discovering issue

[4] BPMN stands for Business Process Modeling Notations, and it is released by OMG's BMI (Business Modeling & Integration) Domain Task Force.

of the workflow performer-activity affiliation networking knowledge from a workflow model. That is, the system's use cases are related with the discovering, analyzing, and visualizing features of the planned knowledge of performer-activity affiliations and allotments. The major use cases and their crucial features are listed as the followings:

- Discovery Use Case : Import XPDL-based workflow models or packages, Discover the wOIS-paan knowledge, and Generate the bipartite matrix from the discovered knowledge
- Analysis Use Case : Calculate the degree centrality of each performer and each activity, and Measure the group-degree centrality of the corresponding workflow models (or packages)
- Visualization Use Case : Visualize the graph nodes and edges between performer and activity in a graphical form of the force-directed-layout of the Prefuse toolkit

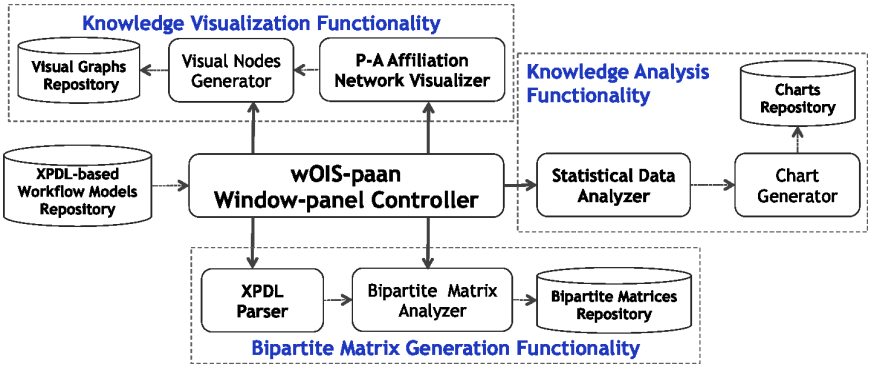

Fig. 1. Architectural Components of the wOIS-paan Knowledge Discovery System

The essential functional components being comprised of the system are bipartite matrix generation functionality[2], knowledge visualization functionality, and knowledge analysis functionality, and these components also can be systematically implemented by using the Java programming language. Fig. 1 illustrates a system architecture of the implemented wOIS-paan knowledge discovery system, which is made up of four groups of architectural components—wOIS-paan Window-control, knowledge visualization, bipartite matrix generation, and knowledge analysis components. Particularly, the XPDL parser of the analysis components group takes charge of generating a performer-activity bipartite matrix from an XPDL-based workflow package[5], and the social graph visualizer of

[5] The system is able to handle a group of XPDL-based workflow models as well as individuals of the workflow models.

the visualization components group depicts the wOIS-paan knowledge as a bipartite graph transformed from the bipartite matrix. In terms of the wOIS-paan knowledge analysis aspect, the system is theoretically backed up by the extended versions of the workload-centrality analysis equations[1], such as actor-degree centrality analysis equations and group-degree centrality analysis equations, so as to mathematically analyze a workflow performer-activity affiliation network model discovered from an XPDL-based workflow package.

2 Significance to the BPM field with a Case Study

As an operational example, we try to discover wOIS-paan knowledge from the XPDL-based pseudo-workflow packages arranged in Table 1. We suppose that there are two pseudo-workflow packages, each of which has two workflow models and three workflow models, respectively, and all fifty activities have been conducted by all of the sixteen performers. Consequently, the system is able to successfully discover a wOIS-paan knowledge from the pseudo-workflow packages, and visualize the discovered knowledge as shown in the captured-screen of Fig. 2. In the visualized wOIS-paan knowledge as colored bipartite graph, boxes and circles imply performers and activities, respectively, and the bold-colored box and its linked circles represent the performer, Alan, and his affiliated 11 activities, such as $\alpha_1, \alpha_9, \alpha_{10}, \alpha_{16}, \alpha_{21}, \alpha_{26}, \alpha_{33}, \alpha_{36}, \alpha_{39}, \alpha_{43}, \alpha_{50}$.

Table 1. Specifications of the XPDL-based Pseudo-workflow Packages

Workflow Package	Workflow Model	Workflow Activity	Workflow Performer
HR-Dept-Workflow-Package1	Hiring-Workflow-Model	$\alpha_1 \sim \alpha_{16}$ (16 Activities)	Jeff, Ed, Christiaan, Emily, Adam, Cynthia,
	Performance-Management-Workflow-Model	$\alpha_{17} \sim \alpha_{25}$ (9 Activities)	Joylette, Amanda, Nathaniel, Bryan, Tamara, Ashley, Ryan, Alan, Chris, Holly
HR-Dept-Workflow-Package2	Employee-Training-Workflow-Model	$\alpha_{26} \sim \alpha_{36}$ (11 Activities)	
	Department-Management-Workflow-Model	$\alpha_{37} \sim \alpha_{44}$ (8 Activities)	The same performers in the Package1
	Salary-Negotiation-Workflow-Model	$\alpha_{45} \sim \alpha_{50}$ (6 Activities)	

3 Conclusion

In this demo-paper, we suggested a possible way of projecting a special affiliation knowledge of the workflow-supported affiliation relations (involvement and participation behaviors) between workflow-based people and workflow-based activities by converging the social network techniques and the workflow discovering techniques. As a consequence of this suggestion, we have newly defined

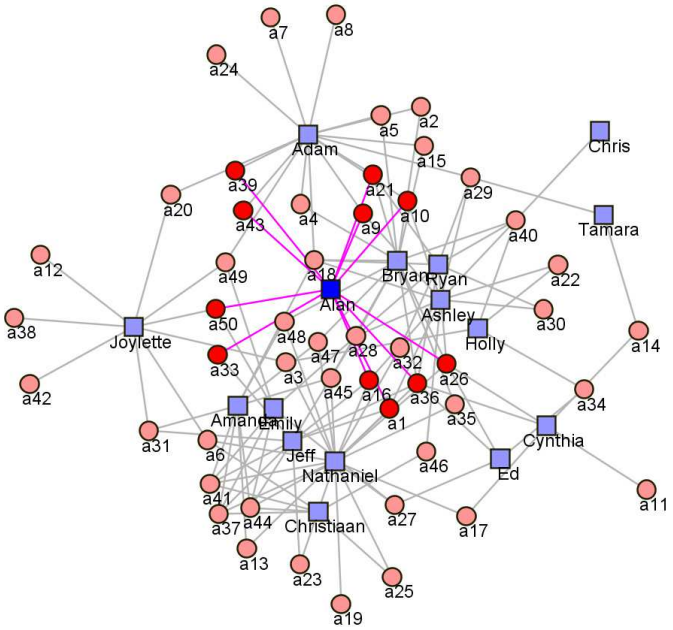

Fig. 2. Visualization of the Discovered wOIS-paan Knowledge by the System

an organizational intelligence of workflow performer-activity affiliation networking knowledge, and implemented a knowledge discovery system to explore the performer-activity affiliation networking knowledge from an XPDL-based workflow package. Conclusively, we have successfully verified the implemented system through applying to two pseudo-workflow packages and visualizing the discovered wOIS-paan knowledge from them.

Acknowledgement. *This research was supported by the Basic Science Research Program (Grant No. 2012006971) through the National Research Foundation of Korea.*

References

1. David Knoke, Song Yang, SOCIAL NETWORK ANALYSIS - 2^{nd} Edition, *Series: Quantitative Applications in the Social Sciences*, SAGE Publications, 2008
2. Haksung Kim, et al.,"A Workflow Affiliation Network Discovery Algorithm," ICIC Express Letters, Vol. 6, No. 3, pp. 765-770, 2011
3. Kwanghoon Kim, "A Workflow-based Social Network Discovery and Analysis System," Proceedings of the International Symposium on Data-driven Process Discovery and Analysis, Campione d'Italia, ITALY, June 29 July 1, 2011
4. Wil M. P. van der Aalst, Hajo A. Reijers, Minseok Song, "Discovering Social Networks from Event Logs," COMPUTER SUPPORTED COOPERATIVE WORK, Vol. 14, No. 6, pp. 549–593, 2005