This is the author's version published as:


Copyright 2011 Elsevier
APROMORE: An Advanced Process Model Repository

Marcello La Rosa\textsuperscript{a,}, Hajo A. Reijers\textsuperscript{b,}, Wil M.P. van der Aalst\textsuperscript{b,}, Remco M. Dijkman\textsuperscript{b,}, Jan Mendling\textsuperscript{c,}, Marlon Dumas\textsuperscript{d,}, Luciano García-Bañuelos\textsuperscript{d,}

\textsuperscript{a}Queensland University of Technology, GPO Box 2434, Brisbane 4001, Australia
\textsuperscript{b}Eindhoven University of Technology, PO Box 513, 5600 MB Eindhoven, The Netherlands
\textsuperscript{c}Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany
\textsuperscript{d}University of Tartu, J Liivi 2, Tartu 50409, Estonia

Abstract

Business process models are becoming available in large numbers due to their widespread use in many industrial applications such as enterprise and quality engineering projects. On the one hand, this raises a challenge as to their proper management: How can it be ensured that the proper process model is always available to the interested stakeholder? On the other hand, the richness of a large set of process models also offers opportunities, for example with respect to the re-use of existing model parts for new models. This paper describes the functionality and architecture of an advanced process model repository, named APROMORE. This tool brings together a rich set of features for the analysis, management and usage of large sets of process models, drawing from state-of-the-art research in the field of process modeling. A prototype of the platform is presented in this paper, demonstrating its feasibility, as well as an outlook on the further development of APROMORE.

Key words: repository, business process model, process collection.

1. Introduction

Business process modeling has become a very popular form of conceptual modeling (Davies et al., 2006). A process model describes, often in some graphical notation, how a certain procedure is composed out of different tasks, which resources are involved in carrying out these tasks, and which objects are being manipulated (Curtis et al., 1992; Giaglis, 2001). One can roughly distinguish between process models that describe procedures as they exist (e.g., to show compliance with quality standards), or that capture alternative ways to produce a particular product or service (e.g., as blueprints for improvement projects). A process model can be used both within the specific context of IT deployment and for more business-oriented purposes (Bandara et al., 2005). Respective examples for these two types of process model usage are the configuration of a workflow management system, and the support of an activity-based calculation of a product’s cost price.

The broad application of process modeling has stimulated contemporary organizations to create dozens, hundreds, and even thousands of process models (Becker et al., 2000; Gulla and Brasethvik, 2000; Reijers et al., 2009; Siegeris and Grasl, 2008). With such massive collections of process models, an apparent issue is how to sensibly deal with these, in particular when considering that models may need to be consulted, updated, and re-used over longer periods of time by various stakeholders. This paper proposes an architecture for an Advanced Process Model Repository – APROMORE – which offers a rich set of features to maintain, analyze, and exploit the content of process models. The features that we envision go well beyond the data-management oriented functionality that is offered by commercial process repositories. Instead, the emphasis is on sophisticated, state-of-the-art features such as advanced model-based analysis, filtering and consolidation. These features may operate on separate and/or sets of related process models.

APROMORE has been implemented as an open-source SaaS (Software-as-a-Service). It is thought to be of interest to practitioners who wish to extract greater value from their process models’ content, technology vendors that wish to extend their offerings by tapping into APROMORE’s features, and researchers who wish to benefit from synergies by incorporating their techniques in the platform and re-using available techniques.

The structure of this paper is as follows. Section 2 provides a background on data repository technologies and specifically on process model repositories. This paves the way for Section 3 and Section 4, which respectively describe the envisioned main features of APROMORE and the service-oriented architecture to support and realize these features. Section 5 presents the internal process definition adopted by APROMORE, which is essential to deal with the multitude of process modeling notations. Section 6 provides a glimpse of the current prototype implementation and describes two typical application scenarios that are supported by this implementation. Finally, Section 7
concludes the paper with a summary and an outlook on future work in this area.

2. Background

This section discusses the background of advanced process model repositories. In Section 2.1 we present concepts and solutions related to general data repositories. In Section 2.2 we then focus on commercial and academic process model repositories.

2.1. Data Repositories

Data repositories have been discussed for quite some time in the database research community. The term repository in this context refers to an extension of a database management system with an explicit control layer with a strong emphasis on metadata management. A repository can then be defined as a “shared database of information about engineered artifacts produced and used by an enterprise” (Bernstein and Dayal, 1994). Database repositories are closely interrelated with the management of static data models. Model management addresses challenges in this area on different levels, from representational questions on a structural level, to processing issues, and topics of organizational embeddedness in the socio-technical system of an enterprise (Dolk and Konsynski, 1984; Blanning, 1993). The main concepts of model management are models and mappings between models (Bernstein, 2003). The major share of mappings can be related to the areas of data warehousing (Jarke et al., 1999) and schema integration (Melnik et al., 2003). Research in all these areas is well-established for static data models, but an overarching approach for process model repositories with integrated model management functionality is still missing these days.

2.2. Process Model Repositories

Process model repositories have been designed both with a focus on workflow execution and on conceptual modeling. In an execution environment, the major focus is on the provision of features for the definition of process control-flow, data structures, resources, and program interfaces (Leymann and Altenhuber, 1994). These main aspects are also present in contemporary BPM tools, but extended e.g. with discovery components for dynamic composition (Weber et al., 2007). In contrast, the focus of conceptual modeling frameworks is more on extension features. For example, IDS Scheer’s ARIS process modeling tool supports the extension of the process metamodel along with customization of symbols (Scheer and Nüttgens, 2000)\(^1\). Standard features include model creation, modification and deletion, accompanied by simple lexical search (e.g. search all models containing ‘Shipment Payment’) and reporting functionality (Lee and Joung, 2000). Similar features are offered by ADONIS (Karagiannis and Kühn, 2002). Other solutions, such as Lombardi’s Blueprint\(^2\) and the ARIS Governance Engine, allow users to visualize changes between multiple versions of a process model. All these systems also offer access control. Specific requirements on a distributed environment including access rights are discussed in (Theling et al., 2005).

The application of semantic technologies has been considered from various angles for process repositories. Work on the commercial tool Semtalk has early recognized the potential of formal ontologies for adding and dynamically changing meta-models in a process repository (Fillies et al., 2003). Formal ontologies are also used in the work of (Klein and Bernstein, 2004). The authors utilize extensive metadata in the process repository for reasoning and process model retrieval. For querying process repositories, they define the Process Query Language (PQL) (Klein and Bernstein, 2004). Repositories also play an important role for an overarching concept of Semantic Business Process Management (SBPM) (Hepp et al., 2005). In (Karastoyanova et al., 2008), the authors identify modeling, system configuration, execution and analysis as key features of a respective SBPM architecture. This architecture builds on three layers: persistence layer, service layer (including locking, version control, and rule inference), and a repository API on top.

3. Capabilities of an advanced process model repository

Besides the standard repository features currently offered by commercial and academic products, such as check-in/check-out, access control, simple search queries, there are several functional areas that can be envisioned when considering advanced support for dealing with process model collections. In this section, we distinguish four areas to discuss such advanced features: i) evaluation, ii) filtering, iii) design, and iv) presentation, and use this classification to propose the capabilities of APROMORE, as reported in Table 1.

As will be shown, this proposal builds on a large set of existing contributions in terms of approaches and techniques which can be adapted to be incorporated in APROMORE. The existence of the various approaches and techniques demonstrates the potential power of APROMORE. It should be noted, however, that each of these contributions focus only on a small piece of the overall landscape of features we envision, and tend to look at process models in isolation, rather than looking at a process model in relation to other models. Moreover, the majority of these approaches and techniques has been devised to work for specific process modeling languages only. Our goal is to provide a wide range of model-independent features that can be directly applied to contemporary process notations such as BPMN, EPCs, YAWL, WS-BPEL, WF-nets, Protos, etc.

3.1. Evaluation

Evaluation is concerned with establishing the adherence of process models to various quality notions.

There is a rich body of knowledge that discusses correctness properties like liveness, boundedness or soundness, mainly based on Petri net concepts (Murata, 1989; Aalst, 1996; Hee et al., 2006). Empirical research has shown that process model
collections from practice typically include a substantial rate of error models (Mendling, 2009). As shown in (Fahlund et al., 2009) there are mature verification approaches available. However, these are rarely supported by commercial tools.

Performance analysis is also a well-established discipline with its roots in operations research and operations management (Anupindi et al., 1999). However, it is often impractical to get meaningful durations, data on execution times and probabilities of alternative branches for performance analysis. Moreover, existing models of human behavior in organizations are too simplistic as demonstrated in (Aalst et al., 2010). Recent research derives simulation parameters from operational systems using process mining techniques (Rozinat et al., 2008). Such features are still missing in commercial tools.

The evaluation of process models has become subject to usability considerations. Research on process model understanding aims to identify the factors that foster or impede model quality (Mendling et al., 2007). Structural metrics such as size or complexity have proven to be closely connected with understanding and error probability (Mendling, 2008). Process modeling guidelines such as the Seven Process Modeling Guidelines (7PMG) (Mendling et al., 2009b) or (Becker et al., 2000) have been proposed. Future tools might directly support them by keeping track of complexity metrics.

### 3.2. Filtering

Filtering offers capabilities to rank the process models in the repository based on their equality or degree of similarity to a query model, or to identify relevant patterns. The heterogeneous representation of comparable behavior has raised the issue of similarity calculation (Dijkman, 2007; Dijkman et al., 2009). In essence, process model similarity determines how close the behavior of two process models is. It can be associated with syntactical, semantical, and contextual aspects of activities in a process model (van Dongen et al., 2008). Taking process behavior into account yields better results than classical metadata based process query techniques such as (Momotko and Subieta, 2004; Klein and Bernstein, 2004), although it is computationally more expensive. Query languages such as BPMN-Q (Awad et al., 2008) can use similarity calculations for ranking query results.

Clone detection allows the identification of duplicate fragments in a process model repository, also known as exact-matching as opposed to approximate-matching used in similarity search. Since sub-graph isomorphism is a computationally expensive operation, indexing techniques need to be employed to compute clones efficiently. Initial work in this direction has been conducted in (Jin et al., 2010; Dumas et al., 2010).

Conformance analysis evaluates to which extent an input model conforms to a reference process model in a given domain. Respective research is discussed in (de Medeiros et al., 2008).

The calculation of similar or exact process model fragments is on a conceptual level closely related to checking whether certain patterns or process fragments are contained in process models (pattern-based analysis). Dedicated query languages such as BPMN-Q or PQL (Klein and Bernstein, 2004) support the formulation of queries to find such patterns.

### 3.3. Design

Design refers to ways to create, modify and complete process models based on existing content.

Merge-driven creation uses merging algorithms to create a new process model from a set of similar models (which can be retrieved via similarity or clone analysis). Related work has discussed process model integration techniques (Aalst and Basten, 2001; Preuner et al., 2001; Grossmann et al., 2005; Mendling and Simon, 2006; Gottschalk et al., 2008), but these techniques are still embryonic and only work for specific process modeling languages. Moreover, commercial tool support is still missing. An exception is made by (La Rosa et al., 2010) where similar process models can be merged into a configurable model which highlights common fragments and distinguishes them from those that are specific to some of the input models.

Pattern-based design allows one to create or complete a process model based on the composition of a set of process fragments (so called “business patterns”) for a specific domain. The general idea of creating a process model based on predefined building blocks has been presented in (Thom et al., 2007).

Individualization relates to the area of configurable process models (Rosemann and Aalst, 2007; La Rosa, 2009). It builds on algorithms to derive a correct process model from a configured process model (Aalst et al., 2010), potentially covering control flow, data and resources involved in a process (La Rosa et al., 2011).

Over time new versions of reference models may be shipped, e.g. resulting from bug-fixes or changes to legislative rules. Similarly, a reference model individualization may also be extended, e.g. to cover customer requirements that are not captured by the reference process model. In this context, extension control is required. It has to establish extension points to control the evolution of configurable reference models and of their individualizations, such that the two types of models can be kept in synchronization. This idea is illustrated in (Balko et al., 2011) but it has not yet been implemented.

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>Filtering</th>
<th>Design</th>
<th>Presentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctness analysis</td>
<td>Similarity search</td>
<td>Merge-driven creation</td>
<td>Abstraction</td>
</tr>
<tr>
<td>Performance analysis</td>
<td>Clone detection</td>
<td>Pattern-based design</td>
<td>Secondary Notation</td>
</tr>
<tr>
<td>Usability analysis</td>
<td>Pattern-based analysis</td>
<td>Individualization</td>
<td>Reporting</td>
</tr>
<tr>
<td></td>
<td>Conformance analysis</td>
<td>Extension control</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Classification of advanced process model repository features according to service areas.
3.4. Presentation

Presentation provides support for improving the understanding of large process models and collections thereof. It relates to useful abstraction mechanisms, secondary notations (color, size, etc.), and reporting facilities.

Abstraction is an important concept to achieve a task-oriented presentation of content for a particular user of a process repository. Different abstraction concepts such as removing infrequent paths and activities and automatically collapsing nodes based on high cohesion/low coupling strategies have been proposed for the simplification and understandability of process models (Günther and Aalst, 2007; Streit et al., 2005; Eshuis and Grefen, 2008; Polyvyanyy et al., 2008).

Secondary notation (Green and Petre, 1996) is a powerful tool to emphasize relevant information without touching the formal structure of a process model. It includes the use of color palettes, e.g. by highlighting the most followed process flow depending on a given user context (Aalst, 2009), icons (Mendling et al., 2009a), and the change of the graph layout. The importance of graph layout is well understood in the conceptual modeling area (Ware et al., 2002; Schreper et al., 2009). Specific layout requirements of BPMN (Object Management Group, 2008) models have been recently discussed in (Kitzmann et al., 2009a; Effinger et al., 2009).

Finally, reporting provides a range of model statics such as number of users or frequency of decisions, to accompany the more traditional visual representation of process models.

The four functional areas that have been discussed up to this point – evaluation, filtering, design and presentation – characterize the main types of features that can be offered by an advanced process model repository. We foresee scenarios where end users will be combining elements from different functional areas. For example, the result of a process model evaluation could lead to an improvement plan describing a number of modifications on the process model (design) to align the latter to a reference model (filtering). Or more, after evaluating the quality of a collection of process models, the best performing models are selected and compared to each other in order to detect similarities (filtering). This result is used to merge the selected models into a configurable reference model (design), which is then presented to the user via a combination of abstraction techniques.

In the next section, we describe the architecture of APROMORE which is tailored towards the above-mentioned features.

4. Architecture

We propose to implement APROMORE via a service-oriented architecture as illustrated in Figure 1. This architecture follows a three-tier model composed of an enterprise layer, an intermediary layer and a basic layer. The enterprise layer is the front-end of the repository. It hosts the repository manager – a public service which exposes the typical amenities of a repository, such as check-in/check-out, simple querying, views, version control, change notification, context management and security (Bernstein and Dayal, 1994). This service is the unique entry point to the repository and can be accessed directly by BPM Suites (BPMS) or reference model vendors for cross-enterprise integration, or via a Web portal by the users of an organization.

The basic layer encapsulates the business logic and data of a traditional software architecture. The business logic consists of the algorithms to operate over process model collections, e.g. matching algorithms, merging algorithms, individualization algorithms. These algorithms are needed to provide the advanced features described in Section 3. Each class of algorithms (evaluation, filtering, design and presentation) is encapsulated by a logic-centric service for reusability and maintainability purposes.

The repository manager accesses these logic-centric services via the batcher service sitting in the intermediary layer. This service acts as a façade over the algorithms and allows users to batch operations via simple scripts that can be submitted through the repository manager. For example, one could search for all models similar to an input process model, merge the result and visualize it in a given notation according to specific abstraction preferences.

The basic layer also hosts a set of data-centric services which serves as an interface to access the underlying persistent data – the core of the repository. Each data-centric service wraps one or more specific data entities and exposes the conventional features of the related DBMS. These include data storage and retrieval, access control, integrity control and transactionality. Five data entities compose this layer:

- models archive: business process models in their original XML formats, e.g. BPMN models in XPDL (Workflow Management Coalition) or EPC models in EPML (Mendling et al.). These can be reference process models for specific domains, individualizations, single models or model collections. Moreover, these models can be configurable (e.g. a configurable reference process model);
- canonical models archive: a canonical representation of each of these models in XML. This canonical format filters out the specificities of a process modeling language, allowing the various repository algorithms to operate on a common process definition (more details in Section 5);
- annotations archive: metadata associated with the canonical models, e.g. layout information for visualization, or search indexes. This metadata is captured in the form of annotations to canonical models and organized in profiles;
- patterns archive: reusable libraries of process definitions for specific industry verticals, defined in canonical format. These can be used, e.g. for model creation, completion or pattern-based evaluation;
- relations archive: the relations between canonical representations of different process models, e.g. relations be-
tween configurable process models and their individualizations, or between process models and their extensions, used for change notification and adaptation control.

The repository manager accesses both process models and their canonical representation via the (de)canonization service – an intermediary adapter equipped with format conversion capabilities. This service is invoked the first time a request is made by the user to check-in a new process model in the repository. This service converts a copy of the model from its original format into its canonical representation; the latter model is indexed by the repository manager and stored in the canonical models archive. From that time onwards, a process model is always accessed through its canonical representation, although it is also persisted in the models archive in its original format (synchronization between the two formats is achieved through metadata). Since the algorithms operate on the canonical format, new models generated through these algorithms, e.g. when merging a set of similar processes into a configurable model, are produced directly in the canonical format. The (de)canonization service is also invoked to convert a canonical representation of a process into some tool or language-specific format when users check-out content from the repository, e.g. when importing a newly created model into a third-party application or into their file system.

5. Canonical Process Format

The canonical process format provides a common, unambiguous representation of business processes captured in different notations and/or at different abstraction levels, such that all process models can be treated alike. The idea behind this format is to represent only the structural characteristics of a process model that are common in the majority of modeling languages. Language-specific concepts are omitted because they cannot be meaningfully interpreted when dealing with process models originating from different notations, i.e. when “cross-language” operations are performed. Moreover, this canonical format is agnostic to graphical information such as shapes, line thickness and positions, which is contained in a concrete process definition. This information is stored separately in the form of annotations, and only used when a canonical model needs to be presented to the user or converted to an original format.

We identify five advantages in using a canonical format for the provision of advanced process model repository features:

1. **Standardization**: a canonical format makes it possible to standardize software access to process definitions via a set of APIs. This is achieved through the (de)canonization adapter, which allows the various algorithms to work on a common process structure. In this way, cross-language operations can be directly performed and concatenated, i.e. without the need to first convert a model into another model’s notation.

2. **Efficiency**: avoiding language conversions in turn improves the overall system efficiency. Moreover, annotations can be used to index canonical elements with specific meanings, with the purpose to expedite queries. In fact, searching large collections for models having particular properties may be very time consuming. Thus having a single optimized format to avoid on-the-fly ad-hoc
conversions is definitely preferable from a performance point of view.

3 \textit{Interchangeability:} annotations also capture non-structural aspects of a process model, such as graphical information or process semantics, which can be automatically inferred from a concrete process definition. By organizing these annotations in profiles, a profile inferred from a process model can be applied to another canonical model, and a canonical model can have multiple profiles. In this way it is e.g. possible to switch between different graphical representations while keeping the same process structure.

4 \textit{Reusability:} the canonical format is also used as the format for storing business process patterns and industry reference models. On the one hand, this facilitates the execution of those operations that involve such content, e.g. conformance analysis or pattern-based completion. On the other hand, it makes this content virtually available in every process modeling language that is supported by the repository.

5 \textit{Flexibility:} the elements of a canonical format are defined through an inheritance mechanism such that at the highest abstraction level a process is simply seen as a directed, attributed graph. This allows algorithms to treat process models at different levels of granularity, depending on the type of operation required by the user.

We observe that without a common process format, a variant of each algorithm would need to be implemented for every (new) process modeling language. Moreover, conversions from one language to another would need to be put in place, to allow cross-language operations such as model comparison and merging.

In the next section we provide a detailed description of the canonical format adopted in APROMORE.

5.1. Metamodel

The metamodel of the canonical process format is defined using the UML class diagram shown in Figure 2. A CanonicalProcess is a container for a set of Nets, ResourceTypes and Objects. Each Net is a directed, attributed graph made up of Nodes and Edges, and represents a process or a subprocess. The top process is indicated as ‘root’, while all other Nets are marked as ‘subnets’. Nodes can be of type Routing or Work, while Edges represent links between Nodes. Routing nodes capture all elements of a process model which are used for routing purposes (i.e. no work is performed from a business perspective), and as such they have more than one incoming edge and/or more than one outgoing edge. They can be Splits (ORSplit for inclusive data-driven choice, XORSplit for exclusive data-driven choice and ANDSplit for parallel branching), Joins (ORJoin for synchronizing merge, XORJoin for simple merge and ANDJoin for synchronization), and States (to indicate the state before an event-driven decision is made or soon after a merge). Splits have one incoming edge and multiple outgoing edges, Joins have multiple incoming edges and one outgoing edge, States can have multiple incoming and outgoing edges. The conditions upon which an (X)ORSplit choice is made, must be specified via the attribute ‘condition’ of each Edge leaving the (X)ORSplit. Also, one such an Edge can be marked as ‘default’ to indicate the default branch to be chosen if the conditions associated with all other Edges leaving the same Split evaluate to false.

Different from Routing nodes, Work nodes capture those elements of a process which are relevant from a business perspective. Work nodes have at most one incoming edge and one outgoing edge and can be partitioned into Tasks and Events. A Task node models a process element which actively performs some work as part of a process, e.g. preparing an invoice or processing a message. Task nodes can be atomic, or compound if they enclose a net describing their behavior. The enclosed net is indicated as ‘subnet’. Events are used to signal the beginning or the end of a process, or to signal something that has happened during a process execution. Events can be specialized into Message events to capture a message being sent or received, and Time events to capture e.g. a timeout or a delay.

Work nodes can be associated with one or more ResourceTypes and Objects. Each ResourceType captures a class of organizational resources participating in the process, i.e. a group of concrete resources rather than the resources themselves. These can be Human, e.g. a position or role in an organization, or Nonhuman, e.g. an information system or equipment. For instance, the Human ResourceType “Finance Officer” may refer to the set of persons of an organization with role Finance Officer. ResourceType can have one or more specializations, e.g. “Finance Officer” may be specialized in “Senior Finance Officer” and “Junior Finance Officer”. This relation is transitive and antisymmetric, and typically indicates a separation of duties. Each association between a Work node and a ResourceType indicates that a resource of that ResourceType is required to carry out the Work node. Therefore, a Work node associated with the same ResourceType \( n \) times, means that \( n \) resources of that ResourceType are required to carry out the given Work node (e.g. this captures the concept of teamwork for human resources, i.e. a set of persons all working on the same task). The association between Work nodes and ResourceTypes can specify a ‘Qualifier’ to indicate the status a given ResourceType takes when performing the associated Work node, e.g. only one person of all the persons with role “Finance Officer” associated with Work node “Prepare invoice” is qualified as “Responsible” person. The association between Work node and ResourceType can be ‘optional’ to indicate that the work may be performed without involving the specific resource (see the attribute optional of ResourceTypeRef).

Objects capture organizational business objects that are involved in the process. These can be physical artifacts, e.g. a paper-based invoice (Hard object) or information artifacts, e.g. a file or variable representing an electronic invoice (Soft object). For the latter, the ‘type’ of the object must be specified, e.g. the file extension or variable type. Objects can be associated with a Work node via an ‘input’ relation if they are utilized by the Work node, and/or via an ‘output’ relation if they
are produced by the Work node. These relations correspond to read/write operations in the case of Soft objects. An object used as both input and output of a Work node indicates that the object is updated, e.g., an invoice is filled-out or a variable changes its content. Moreover, input objects can be marked as ‘consumed’ if they are destroyed while being used by a Work node. Similar to ResourceTypes, the association between Objects and Work nodes can also be tagged ‘optional’ to capture a situation where the Work node may be performed without using or producing the specific object.

Nodes, ResourceTypes and Objects can be configurable. This is denoted by their optional attribute ‘configurable’. A node’s configuration options are indicated through annotations outside a canonical representation. Configuration is an important aspect for large model repositories. However, given the diversity of languages and concepts, the configuration mechanism itself is not part of the canonical process format.

In the next section we motivate the choice for such elements and show how these are mapped to elements in concrete process modeling languages.

5.2. Methodology and Mapping

The elements to be included in the canonical format were identified from an analysis of commonalities among six widely adopted business process modeling languages. These are two languages for conceptual process modeling: EPCs (Keller et al., 1992) and BPMN 1.2 (Object Management Group, 2008), and four languages for executable process modeling: Protos 8.0, WF-Nets (Aalst, 1998), YAWL 2.0 (Aalst and Hofstede, 2005) and WS-BPEL 2.0 (OASIS, 2007).

To date, EPCs (Event-driven Process Chains) are probably the most used process modeling notation among practitioners. Initially developed for the design of the SAP R/3 reference process model (Curran and Keller, 1997), they later became the core modeling language of the ARIS platform, and were adopted by other vendors for the design of SAP-independent reference models (e.g. the ARIS-based reference models for ITIL (IDS Scheer, 2009a) or SCOR (IDS Scheer, 2009b)). We support EPCs along with two extensions: eEPCs (extended EPCs) (Scheer, 1999) and iEPCs (integrated EPCs) (La Rosa et al., 2011). These cater for the representation of organizational resources and objects participating in a process, and for the representation of variation points on top of these elements respectively.

BPMN (Business Process Modeling Notation) is an emerging notation that can be seen as alternative to EPCs. BPMN was designed with the intent to enable both business users and technical developers to model readily understandable graphical representations of business processes. The BPMN specification is driven by the OMG (Object Management Group) standardization committee and is supported by a growing number of organizations and IT vendors.

WF-Nets (Workflow nets) are a class of Petri nets specifically designed for modeling executable business processes (Aalst, 1998). As such, they benefit from a rich body of theoretical results, analysis techniques and tools (Murata, 1989). WF-Nets have been extensively applied in academia to the formal
verification of business process models (Verbeek et al., 2001). Specifically, we chose to adopt the core WF-Net notation which does not feature triggers, explicit splits and joins, and hierarchical transitions, to avoid overlaps with the YAWL language.

YAWL (Yet Another Workflow Language) is an expressive language to describe, analyze and automate complex business process specifications, which builds on top of WF-Nets and provides comprehensive support for the Workflow Patterns (Aalst et al., 2003). YAWL is widely used in teaching and research, and is facing an increased uptake in industry (YAWL Foundation).

Finally, WS-BPEL (Web Service Business Process Execution Language) is an alternative to YAWL, which describes business process models as a composition of Web services. For this reason, BPEL represents a convergence between Web services and business process technology. Its specification derives from the joint effort of a number of IT vendors and has been standardized by the OASIS (OASIS, 2009) consortium.

We compared all modeling elements in the above languages with each other, by looking at the underlying concepts captured by each element. For example, EPC functions were compared with BPMN transitions, Protos activities, YAWL tasks and BPEL activities, since they all capture the concept of “performing some work”. In order to do so, we first decomposed any concrete language construct in its fundamental concepts. For example, in YAWL splits and joins are always attached to tasks, so we extrapolated the join and split behavior from a YAWL task, and compared the former two with the routing elements of the other languages (more details on the conversion of language constructs are provided below). From this analysis, we created a canonical element for each concept that was shared by at least four languages out of the six taken into account.

Table 2 illustrates how the canonical elements from Figure 2 are mapped to concrete modeling elements. We can observe that basic concepts such as ‘Edge’, ‘Task’, ‘Event’ and ‘ANDSplit/Join’ are shared by all languages, others such as ‘State’, ‘XORSplit/Join’ and ‘Object’ are only shared by five languages, while more advanced concepts such as ‘OR-Split/Join’, ‘ ResourceType’ and the specific Event types ‘Message’ and ‘Timer’, are only supported by four out of six languages. The table uses the term ‘sese’ (single-entry single-exit) to refer to model elements with one incoming and one outgoing arc. These elements are treated different from nodes where a flow splits or multiple flows join (more details are provided below).

Table 2 also lists at the bottom those elements that are not supported by the canonical format. These elements refer to concepts that are not sufficiently represented in the six languages examined, such as error handling, cancelations or multiple instance tasks. Hence, supporting these concepts would have led to canonical elements being too language-specific. Moreover, these concepts are typically not interpreted by the various algorithms available in the literature. Nonetheless, the canonical format can be extended without varying its core structure, should new concepts be needed in future.

Figure 3 concludes the discussion on the canonical format by illustrating the canonical representation of thirteen common language constructs, according to the mapping in Table 2. Each construct (central column) is converted into a directed, attributed graph (right column) made up of a number of Nodes and Edges which are annotated with a reference to the respective element in the concrete language (trivial Edge annotations are omitted).

The first construct is taken from EPCs and represents a data-driven decision. EPCs only provide three modeling elements to capture a process control-flow: Functions, Connectors and Events. Functions are always mapped to Tasks while Connectors are mapped to Splits and Joins, depending on their type. Events are mapped to canonical Events if they do not immediately follow an (X)OR-split Connector, otherwise to Edges. This is because in EPCs the Events following an (X)OR-split are actually used to represent the conditions upon which the (X)ORSplit choice is made. Therefore, in this example the two EPC Events in the example of Figure 3 are mapped to two Edges, and their labels to the attribute ‘condition’ of these Edges.

The second and third constructs are two examples of syntactic sugar offered by BPMN, i.e. more concise representations of a given concept. The two incoming Flows to Task A in the second construct, are a compact notation for an Exclusive-join Gateway, while the two outgoing Flows are a compact notation for a Parallel-split Gateway. This translates to a canonical graph with one Work Node preceded by an XORJoin Node (mapping the implicit join) and followed by an ANDSplit Node (mapping the implicit split). Similarly, the third construct shows the compact notation for an Inclusive-split Gateway via two Conditional Flows (each capturing a condition for the choice) and one Default Flow (capturing the default condition). The corresponding canonical graph will have an ORSplit Node to capture the inclusive choice and one Edge for each outgoing Flow, with the Default Flow being mapped into an Edge with attribute ‘default’ set to true.

The fourth construct shows how AND-join and AND-split are modeled in WF-Nets and Protos. This behavior is never explicitly represented, but captured via incoming/outgoing Flows (or Connections in the case of Protos) to/from a Transition (called Activity in Protos). Furthermore, Protos allows one to specify the (X)OR-join and -split behavior for an Activity. Again, this is not explicitly shown and must be specified through an Activity’s properties (fifth construct in Figure 3). In both cases, we need to add two extra Routing Nodes in the canonical representation to explicitly capture the split and join behavior, besides a Work Node to capture the Transition or Activity.

Unlike some of the other languages, in YAWL splits and joins must be explicitly represented as Task decorations (sixth construct in Figure 3). This is because they are semantically
<table>
<thead>
<tr>
<th>Canonical element</th>
<th>(e)iEPC</th>
<th>BPMN 1.1</th>
<th>WF-Net</th>
<th>Protos 8.0</th>
<th>YAWL 2.0</th>
<th>WS-BPEL 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net</td>
<td>EPC, Compound Function (Sub)Process</td>
<td>(Sub)Process</td>
<td>(Sub)Process</td>
<td>Process, Scope</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Edge</td>
<td>Arc, Event subsequent to (X)OR-split Connector</td>
<td>Sequence Flow, Conditional Flow, Default Flow</td>
<td>Arc</td>
<td>Connection</td>
<td>Flow</td>
<td>Sequence, Link</td>
</tr>
<tr>
<td>Task</td>
<td>Function</td>
<td>Task</td>
<td>Transition</td>
<td>Activity</td>
<td>Task</td>
<td>Assign, Empty, Extension/Activity</td>
</tr>
<tr>
<td>Event</td>
<td>Event not subsequent to (X)OR-split Connector</td>
<td>Plain Event, Start Rule Event</td>
<td>Input Place, Output Place, sese Place</td>
<td>sese Status</td>
<td>Input Condition, Output Condition, sese Condition</td>
<td>[mapped directly to specific types Message and Timer]</td>
</tr>
<tr>
<td>Message Event</td>
<td>Message Event</td>
<td>Message Event</td>
<td>any Trigger except Time</td>
<td>WSInvoker Task</td>
<td>Invoke, Receive, Reply, onMessage in Pick</td>
<td></td>
</tr>
<tr>
<td>Timer Event</td>
<td>Timer Event</td>
<td>Time Trigger</td>
<td>Timer Task</td>
<td>Wait, on Alarm in Pick</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANDSplit</td>
<td>AND-split Connector</td>
<td>Parallel-split Gateway, Task’s Parallel-split</td>
<td>Transition’s AND-split</td>
<td>Activity’s OR-split</td>
<td>Task’s OR-split</td>
<td>Flow (opening tag), Source Link</td>
</tr>
<tr>
<td>ORSplit</td>
<td>OR-split Connector</td>
<td>Inclusive-split Gateway, Task’s Inclusive-split</td>
<td>Activity’s OR-split</td>
<td>Task’s OR-split</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XORSplit</td>
<td>XOR-split Connector</td>
<td>Data-based Exclusive-split Gateway</td>
<td>Activity’s XOR-split</td>
<td>Task’s XOR-split</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANDJoin</td>
<td>AND-join Connector</td>
<td>Parallel-join Gateway</td>
<td>Transition’s AND-join</td>
<td>Activity’s AND-join</td>
<td>Task’s AND-join</td>
<td>Flow (closing tag), Target Link</td>
</tr>
<tr>
<td>ORJoin</td>
<td>OR-join Connector</td>
<td>Inclusive-join Gateway</td>
<td>Activity’s OR-join</td>
<td>Task’s OR-join</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XORJoin</td>
<td>XOR-join Connector</td>
<td>Exclusive-join Gateway, Task’s Exclusive-join</td>
<td>Activity’s XOR-join</td>
<td>Task’s XOR-join</td>
<td>If (closing tag), While (closing tag), RepeatUntil (closing tag)</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>Event-based Exclusive-split Gateway</td>
<td>non-sese (Input/Output) Place</td>
<td>non-sese Status</td>
<td>non-sese (Input/Output) Condition</td>
<td>Pick (opening tag)</td>
<td></td>
</tr>
<tr>
<td>ResourceType (Human/Nonhuman)</td>
<td>Org. Unit (eEPC), Position (eEPC), Supporting System (eEPC), Role (iEPC)</td>
<td>Pool, Lane</td>
<td>Role, Role Group, Responsible (resource), Application</td>
<td>Role, Org. Group, Position, Capability Custom Service</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object (Hard/Soft)</td>
<td>Object (eEPC)</td>
<td>Data Object</td>
<td>Document, Folder, Data Group, Data element</td>
<td>Task Variable</td>
<td>Variable, For and Until in Wait and onAlarm</td>
<td></td>
</tr>
<tr>
<td>ProcessInterface, Person (eEPC)</td>
<td>Terminate, Complex Gateway, Events: Link, Error, Multiple, Compensation, Cancel, Signal, Events on Task boundary, Multiple Instance, Block repetition, Adhoc, Transaction, Message Flow, Exception Flow</td>
<td>Multiple, Buffer, Team, Batch</td>
<td>Multiple Instance, Participant</td>
<td>Exit, ForEach, Throw, Validate, Handlers, Correlations</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Conversion chart for the canonical format, including concrete elements not supported (sese = single-entry single-exit).
Figure 3: Canonical representation of common concrete language constructs.

bound to the Task’s behavior. Therefore in the canonical format we need to separate a Task from its routing behavior.

A Place in WF-Net, a Status in Protos and a Condition in YAWL, are all represented by a circle in the respective notations, and used to capture either a state of the process or an event occurring during the execution of a process. In the canonical format we separate these two concepts: if the Place (Status, Condition) is sese, we map it to an Event, whereas if it is non-sese, we map it to a State. This latter situation is shown by the seventh construct in Figure 3, where there are two incoming and two outgoing arcs.

A Place (Condition) is also used to capture the beginning/end of a WF-Net (YAWL) process. In this case we map the Place (Condition) to an Event irrespective of the number of incoming/outgoing arcs. However, if the Place (Condition) is non-sese, we also map it to a State. This can occur when an Input Place (Condition) has more than one outgoing arc or when an Output Place (Condition) has more than one incoming arc. In the first case, the added State is used to capture the state in which an event-driven decision is taken; in the second case, it is used to capture the final state before the process ends. The eight construct shows the canonical representation of a non-sese Input Place (Input Condition), while the ninth construct shows that of a non-sese Output Place (Output Condition).

The last four constructs show the canonical representation of the routing activities of WS-BPEL. If (to model exclusive data-driven choices), While and RepeatUntil (to model loops) and Flow (to model parallelism). The WS-BPEL standard does not define a standard visual representation for its processes, therefore we used the WS-BPEL XML format to illustrate these examples.

6. Prototype Implementation

As a proof of concept, we implemented a prototype of APROMORE according to the architecture described in Section 4.4 The prototype supports the following features:

- **Basic features:** model import/export, model search, model classification;
- **Filtering functionality:** similarity search;
- **Design functionality:** merge-driven creation.

These features can be accessed via a Web portal, or directly by using the available Web services. The portal exposes the above features through a graphical interface to provide process models visualization and editing capabilities (see Figure 4). Specifically, the portal is implemented using the so-called “model-view-controller” pattern, where the portal itself is merely a view on the models stored in the underlying database with Java methods acting as controllers. The algorithms for similarity search and harmonization-driven creation,
as well as the (de)canonization adapter, are exposed as Web services through a standard WSDL interface.

Internally, both the models archive and the canonical models archive are implemented using a single MySQL database, although these are exposed as two separate logical entities through data-centric Web services. Currently, the process modeling languages that are supported are EPCs and BPMN.

In the following we describe two example scenarios that are supported by the prototype repository.

In the first scenario an organization can use the business process repository for advanced search functionality. The collection of process models to be analyzed can originate from some proprietary BPMS and then imported into the repository. Depending on the features provided by the external BPMS and the level of integration between the latter and the repository, BPMS users can profit from the advanced features provided by the repository. For example, they can search for a particular model based on keywords, on models classification (e.g., per industry vertical) or by using an input model that is similar to the model to be searched for. A tighter integration is possible by invoking the Web services provided by the repository directly from the BPMS. Alternatively, a link is established between the repository Web-portal and the BPMS such that users can use the Web-portal to perform their searches and upon opening a process model in the portal, the BPMS is launched.

In the second scenario the harmonization-driven creation provided by the repository can be used to assist an organization integrating their process models with those of another organization, e.g., as part of a merger or acquisition. Both organizations can import their sets of business processes in the repository (provided that they are in a format that the repository supports). Subsequently, they can use the similarity search function to search for pairs of models that are similar. In a next step they can be aided in establishing a match between elements from one process and elements from the other process and merge the two models into a configurable process model, using this match. The resulting model will capture both the commonalities between the two models, and their differences, in the form of variation points. This new model can then provide a roadmap for implementing changes to the current business services and IT infrastructure supporting the business process, in order to rationalize them.

7. Conclusion

This paper presents APROMORE, which is an advanced repository to hold, analyze and re-use large sets of process models. APROMORE is an open source platform implemented according to the principles of service-oriented architectures, and exposed to the end user via a Web interface. The canonical format for process modeling notations is an essential ingredient for dealing with the diversity of available notations. A prototype demonstrating the feasibility of APROMORE is also presented in this paper.

The contribution of APROMORE can be discussed from two angles. Considering the interests of practitioners, the tool is thought to be helpful in dealing with many of the challenges that stakeholders face when dealing with large numbers of process models. In this respect, APROMORE goes well beyond the typical capabilities offered by commercial tools, e.g., basic access control and simple version control. APROMORE provides advanced support for dealing with models in different notations, platform-independent access, and maintaining the relations that exist between models. Also, APROMORE is capable of incorporating a collection of state-of-the-art techniques for analyzing, visualizing, transforming, and creating process models, which hitherto have mostly been known to the research community but hardly used in practice. In this way, APROMORE can be regarded as a tangible means for knowledge transfer to the process modeling praxis.

From a research perspective, it is noteworthy that APROMORE is open to all researchers who have an interest in applying and developing techniques dealing with the analysis and optimization of process models. Due to its service-oriented architecture, it will be relatively easy for researchers to develop their own services and Web plug-ins, offering new capabilities while interacting with existing ones, by relying on the common infrastructure of APROMORE. In this way, APROMORE offers a separation of concerns that we hope is appreciated by our fellow researchers. An initiative that inspired us in this respect is ProM, which is highly successful as an open research platform in the field of process mining.

At present, one of the challenges is to populate the repository with large sets of process models to be used as a test-bed and which can help to unleash the capabilities of APROMORE. Thanks to the involvement of four research groups in the APROMORE project, we have some 2000+ models available. Most of them are of medium size (20-100 tasks) and have

5  www.processmining.org
been developed either in an academic or in an industrial context. At this stage, we are in touch with various industrial partners in the financial, healthcare, governmental, and creative industries domains to involve them in this initiative. Other challenges relate to more technical and operational issues, such as adding open, Web-based interfaces to proprietary implementations of analysis techniques, ensuring that the hardware can scale with the use in APROMORE, aligning with access models such as the OpenId initiative,6 and integrating with other open platforms such as ProM and the Oryx visual editor7.

With respect to future research, our efforts will be devoted to the development of new analysis and management techniques to be integrated in APROMORE. One example would be the development of an advanced version control system that can provide a semi-automatic resolution of conflicting process model updates. This is a relevant characteristic for a modern collaborative process modeling environment where it is realistic to assume that many stakeholders with different skills and responsibilities will partake in the modeling activity, thus potentially generating conflicting versions that need to be harmonized.

In conclusion, we believe that APROMORE is an important step in reaching a more mature level of dealing with the management of massive collections of process models. This entails challenges, but is highly relevant for practice, and serves as an exciting area for research. We hope that both practitioners and researchers will join us in the further development of APROMORE.

References


---

6http://openid.net
7http://bpt.hpi.uni-potsdam.de/Oryx


