Process Discovery from Model and Text Artefacts (Preprint)

Aditya Ghose, George Koliadis, Arthur Chueng
Decision Systems Lab (DSL)
School of Computer Science and Software Engineering
University of Wollongong Wollongong, N.S.W. Australia

Abstract

Modeling is an important and time consuming part of the Business Process Management life-cycle. An analyst reviews existing documentation and queries relevant domain experts to construct both mental and concrete models of the domain. To aid this exercise, we propose the Rapid Business Process Discovery (R-BPD) framework and prototype tool that can query heterogeneous information resources (e.g. corporate documentation, web-content, code e.t.c.) and rapidly construct proto-models to be incrementally adjusted to correctness by an analyst. This constitutes a departure from building and constructing models toward just editing them. We believe this rapid mixed-initiative modeling will increase analyst productivity by significant orders of magnitude over traditional approaches. Furthermore, the possibility of using the approach in distributed and real-time settings seems appealing and may help in significantly improving the quality of the models being developed w.r.t. being consistent, complete, and concise.

1 Introduction

Modeling is an important, expensive, time-consuming and labour-intensive part of the business process lifecycle. The first major step in managing a business process is discovery (or understanding) [19] [23]), with subsequent improvement initiatives driven by a need to “understand existing processes and evolve these processes in ways that maintain their strengths” [21]. In this paper we report on a project to build a tool-kit that would shift the focus in modeling from model-building to model-editing. Our aim is to address the model acquisition bottleneck, a version of the well-known knowledge acquisition bottleneck [9]. Our guiding premise is that most organizations maintain enterprise repositories of (sometimes legacy) documents and models which can provide rich sources of information that could be mined to extract “first-cut” process models. Our premise is also that by extracting such “first-cut” models (henceforth referred to as proto-models) and presenting them to an analyst for editing, such a toolkit can significantly enhance analyst productivity. Given that organizations are often loathe to invest the resources required for significant modeling exercises, the availability of such a toolkit can make modeling-in-the-large a viable option in many instances.

We classify the artefacts (henceforth called source artefacts) typically available in an enterprise repository into two categories: text and model. Text artefacts are documents such as memos, manuals, requirements documents, design documents, mission/vision statements, meeting minutes etc. Model artefacts could be models in a variety of notations, including UML design models, or enterprise models or rule models. We define two categories of model extraction techniques: text-to-model extraction (for extracting process models from text artefacts) and model-to-model extraction (for extracting process models from models in other notations). We describe the R-BPD (Rapid Business Process Discovery) toolkit in which instances of these extraction techniques have been implemented. Additional R-BPD examples and code fragments that could not be included in this paper due to space constraints are available at the following site: www.uow.edu.au/~aditya/projects/model-discovery.html.

The R-BPD can potentially extract a large number of (sometimes small) process proto-models from an enterprise repository. Some of these proto-models might in fact be alternative descriptions of the same process. We describe heuristic techniques for establishing model identity to deal with such situations. When multiple models that seem to describe the same process are identified, we need to cross-validate these against each other. We use model consistency as a basis for cross-validation, i.e., alternative consistent descriptions of the same process are viewed as supporting each other. We define a lightweight structural check for consistency and provide examples. Finally, we describe how the R-BPD toolkit can also support traceability and change management.

Our research relies, for its conceptual foundations, on several other areas of inquiry. Knowledge acquisition and
modeling “can be the most time consuming portion of the knowledge engineering process” [3], and has been termed as the “bottleneck” in expert systems design [11]. This issue covers the inherent difficulties (inc. time and resources required) in eliciting complete and concise knowledge from experts.

Some of the major considerations include:

- the *choice* of approach (or combination) to use for acquiring specific types of knowledge, as captured by the differential access hypothesis [11] [26];
- the tradeoff between the *acquirability* (i.e. usability by a particular audience) and *expressive power* (or applicability) of languages used to represent knowledge [3], which may be reduced via the combined use of multiple languages and views on a domain of interest;
- and, *bias* that “is the result of cognitive heuristics” and is mostly introduced when: individual or group experience is overestimated; there is inappropriate emphasis on specific phenomena; small sample sizes are used; there is over-confidence in levels of certainty; or, there is over-estimation of data completeness [18].

Automated knowledge acquisition shells provide an interesting approach toward a solution. Such shells iteratively construct and propose queries to experts to discover computable representations of the domain [10]. In particular, the Requirements Apprentice [17] project contributes a system (as a mediator) for applying knowledge acquisition techniques to support the “transition between informal and formal specifications...” [17]. The apprentice “avoids involvement with the surface syntax of natural language [specifications]” [17] to ensure the “deeper” problems in natural language understanding are not encountered.

A growing body of literature has been established on Workflow Mining [5]. These approaches focus on extracting meaningful process models by analyzing event logs that are generated by transactional information systems. They have also been extended to mine social networks [22] and decision junctions [1]. In [8], an extended scope for workflow mining is proposed, which allows for the discovery of activities and social phenomena in combination with traditional approaches that capture precedence relationships between activities. Activities are induced from similar sets of ordered database transactions. Once activities are discovered, traditional mining techniques are applied to business process traces to discovery precedence relationships. In addition, the approach used to induce activities is applied to the discovery of roles by analyzing the similarity between the behaviors of actors.


Model Driven Architecture (MDA) is primarily concerned with the automated transformation of models from abstract domain descriptions directly into implementable solutions (see [4] for a taxonomy of approaches). Our approach for model-to-model translation (see Section 3) extends and partially automates the constrained development method we presented in [14] for managing business process model (BPMN, see Figure 4) lifecycles with organizational models (i.e., see Figure 3). In essence, [14] described an methodology to managing process lifecycles with explicit high-level organizational models by guiding the derivation or maintenance of one type of model given the availability of the other. This was achieved by: 1/ Establishing a correspondence between elements within the models that describe aspects of the process; 2/ Annotating sets of elements common to both models with semantics in natural language (with the intent of automated translation into structured, formal notation); 3/ Applying specific rules and procedures for determining consistency between such models; and, 4/ Using the results of the consistency check to help guide refinement and co-evolution of the models toward correctness.

Figure 1. Enterprise Process Life-cycle Manager (EPLM) Prototype

To date, we have also developed the Enterprise Process Lifecycle Management (EPLM) toolkit (see Figure 1) above a commercial CASE platform [12] for experimentation. The EPLM partially automates the procedures described in [14]. This paper extends [14] by contributing a means to rapidly extract and identify partial proto-process
models from a variety of sources and incrementally combine them toward completeness and correctness.

In [25], an approach for discovery is described, which proposes a three-layered language for representing business processes and the use of model-checking techniques for verifying constructed models against business requirements that are specified a-priori. They presume the correctness of the requirements provided and cannot resolve inconsistencies, manage change and evaluate correctness within and across the inter-relationships between models constructed in a such a variety of languages.

2 Text-to-Model Extraction

To support process discovery, we employ both text-to-model and model-to-model translation as follows. The examples presented follow from an R-BPDTk test case based on a tutorial provided by BEA Systems [2].

The intent of our approach for text-to-model extraction is to look for cues within text documents that suggest “snippets” of process models. In the R-BPDTk, we use two sets of techniques for text-to-model extraction:

- **Template-Based Extraction:** Here we construct templates of commonly occurring textual cues for processes, and extract proto-models by scanning documents for instances of these patterns. For example, one such cue that we currently use in the R-BPDTk simply extracts sentences that conform to the pattern: “If <condition/event>, [then] <action>.”

  For example. “If the credit check fails, the customer service representative is assigned the task of notifying the customer to obtain correct credit information, and the process becomes manual from this point on.”

- **Information extraction-based:** Here we use NLP toolkits such as NLTk [13] to extract verb phrases, verbs, temporal connectives etc. as the building blocks for process models.

  For example we can extract noun phrases (np) such as “the customer”, verb phrases (vp) such as “notifying”, and extract possible activities (a) by looking for <vp, np> pairs or possible role/assignments from <np, a> pairs where np refers to an actor.

  Both these approaches provide a rapid means to extract, interpret and summarize the knowledge contained within text documents. This is not to say that the machine interpretation will always be valid, however analyst support and automated identification and cross-validation functions will help to unearth inconsistencies (e.g. where the np in an assignment pair is found to be an inanimate object rather than an actor - also see Section 4). In addition, other advanced and efficient methods are available that may help in reducing errors, for example when interpreting temporal relationships [16].

The following fragment shows output from the R-BPD tool that assigns an activity to a role. The activity in question is “check customer credit information” and the role in question is that of “customer service representative”. A combination of template-based extraction and analysis using the NLTk toolkit is used to identify both the role and its relationship to the activity.

```
{assignment: (np: ('a', 'at') ('customer', 'nn') ('service', 'nn') ('representative', 'nn')) ('who', 'wpo') (activity: (vp: ('checks', 'vbz')) (np: ('the', 'at') ('customer’s', 'nn$') ('credit', 'nn') ('information', 'nn')))}
```

This second fragment indicates a precedence relationship between two activities - specifically that “order cancellation” may only occur prior to “shipping”.

```
{rule: ('before', 'cs') (activity: (vp: ('shipping', 'vbg'))) ('', '')}
```
3 Model-to-Model Extraction

The intent of model-to-model extraction is to obtain elements of process models that might be described in other existing models. For instance, a sequence diagram can be viewed as a fine-grained proto-process model. We can conceive of the following two alternative approaches for model-to-model extraction:

- **Syntactic mappings between notations:** In this approach, hand-crafted mapping functions are used to map models in a variety of different notations (e.g., use case diagrams, sequence diagrams, state diagrams etc.) to a process modeling notation. Our current work is primarily based on this approach. Let \( N_i \) and \( N_j \) be two distinct modeling notations. Let \( f_{\text{syn}}^{N_i,N_j} : M_{N_i} \rightarrow M_{N_j} \) where \( M_{N_i} \) and \( M_{N_j} \) are the sets of all possible models expressible in \( N_i \) and \( N_j \) respectively, be a function that maps a model in \( N_i \) to a model in \( N_j \). That is, the function generates an \( N_j \) model that expresses as much of the input model (in \( N_i \)) as can be expressed in \( N_j \). We shall refer to such functions as syntactic transformation functions and note that such functions can be realized using QVT languages in the model-driven architectures framework (although our current implementation does not use a QVT language).

- **Mappings based on semantic correspondences:** Here we would rely on semantic correspondences that exist between languages to establish what statements can be said in another language, and how they should be represented syntactically. In this case, analyst involvement will mainly be required where either \( N_i \) or \( N_j \) is an informal notation. Where the semantics for both \( N_i \) and \( N_j \) are well-understood, we would require a function \( f_{\text{sem}}^{N_i,N_j} : \text{Sem}_{N_i} \rightarrow \text{Sem}_{N_j} \), where \( \text{Sem}_{N_i} \) and \( \text{Sem}_{N_j} \) represent the semantic domains of \( N_i \) and \( N_j \) respectively. Our current implementation does not adopt this approach but [14] provides some preliminary indications of how an analyst-driven “mixed” approach might look like.

The following describes a syntactic transformation function that maps \( i^* \) Strategic Rationale (SR) models [27] to BPMN proto-models:

1. Represent each activity within an \( i^* \) model as a BPMN process model.

2. Traverse the internal goal graph of each actor within an \( i^* \) model in a parent-child manner, establishing the following relations among mapped BPMN process models.

   (a) For each AND task decomposition of a parent task in the \( i^* \) model, include a representation of the decomposed tasks mapped in BPMN as sub-processes within the parent BPMN process model.

   (b) For each task represented within a mapped BPMN process model, assign the task to the responsible actor assigned the task by placing it within a pool taking on the \( i^* \) actor label.

3. Traverse the dependencies between actors within the \( i^* \) model, establishing the following relations among mapped BPMN process models.

   (a) For each task on the depender side of a dependency, include the mapped BPMN model as part of the BPMN process model, assign the task to the responsible actor assigned the task by placing it within a pool taking on the depender’s actor name.

   (b) For each resource dependency, include a message flow link between associated tasks within each BPMN model, whereby the source of the message flow maps to the dependee’s task and the target to the depender’s. Label the message with the resource name. In the case whereby an actor within the \( i^* \) model is represented in Strategic Dependency (SD) model, associate the message flow link to that actor’s pool boundary.

Figure 4 is an example of a BPMN model thus extracted from a \( i^* \) model (depicted in Figure 3).

As another instance of a syntactic transformation function, consider the UML Interaction Diagram in Figure 5, mapped to the BPMN process model [24] in Figure 6. In this case, the function \( f_{\text{ID,BP}}^{\text{ppm}} \) mapped: objects in an interaction diagram to pools within the BPMN process model; and, interactions to messages and activities within the BPMN model. The BPMN model may then be easily edited to refine sequencing information and additional activities, that may also trigger some change in the interaction diagram if a reverse mapping were applied.
Distinct process models obtained via model-to-model extraction and text-to-model extraction might actually describe the same process. This presents both an opportunity and a challenge. The opportunity lies in the ability to cross-validate distinct models of the same process against each other. The challenge relates to the problem of establishing model identity, i.e., determining whether two distinct process models refer to the same process.

Determining model identity is a difficult problem. The R-BPD tool generates proto-models from multiple sources. Consider a process model $m_1$ generated from one or more text documents and process model $m_2$ extracted from one or more legacy UML Interaction Diagrams. $m_1$ and $m_2$ might indeed be alternative descriptions of the same process, but determining whether this is the case is the problem of model identity. One approach would be to relegate that decision to the analyst. It would, however, be useful if the tool were able to establish tentative identity relationships between distinct process models and use these to perform cross-validation across models (discussed below).

The process of establishing model identity can be decomposed into three steps. First, we need to resolve naming conflicts (i.e., the use of distinct names for the same concept - “shipment” and “consignment” for instance). We use an enterprise ontology for this purpose. Second, we need to resolve abstraction conflicts, which relates to the problem of describing the same process at varying levels of abstraction. Here too we require an enterprise ontology. Within such an ontology, background rules such as $\text{Performs}_1(\text{ProcessingSystem}, \text{Read}, \text{Order}) \land \text{Performs}_2(\text{ProcessingSystem}, \text{AppendID}, \text{Order}) \Rightarrow \text{Performs}_3(\text{ProcessingSystem}, \text{Receive}, \text{Order})$ permit us to relate finer-grained descriptions of a process (containing, say, $\text{Step}_1$ and $\text{Step}_2$) with more abstract descriptions of the same process (containing $\text{Step}_3$) (this rule corresponds to the natural language statement “The order is received by a processing system, which reads the data and appends an ID number to the order.”).

Having resolved naming conflicts and having established relationships between descriptions at different levels of abstraction, the third step involves actually establishing whether two process models indeed describe the same process. In general terms, this involves devising a similarity function that takes as input a pair of process models and produces as output a similarity measure. If the similarity measure meets a pre-specified threshold, the two models are deemed to describe the same process. In the current implementation of the R-BPD tool, we use a simple structural similarity function that exploits a graph encoding of BPMN models (our approach resembles that of [15] in some respects). In the resulting digraph $(V,E)$, each node is of the form $<ID, \text{nodetype}, \text{owner}>$ and each edge is of the form $<u,v, \text{edgetype}>$. Each event, activity or gateway in a BPMN model maps to a node, with the $\text{nodetype}$ indicating whether the node was obtained from an event, activity or gateway respectively in the BPMN model.
of nodes of type event or activity refers to the ID of the corresponding event or activity in the BPMN model. The ID of a gateway type node refers to the condition associated with the corresponding gateway in the BPMN model. The owner attribute of a node refers to the role associated with the pool from which the node was obtained. The edgetype of an edge can be either control or message depending on whether the edge represents a control flow or message flow in the BPMN model. To obtain a similarity measure between two process models, we first encode the process models into digraphs $d_i$ and $d_j$ as described above. We then compute the total number of nodes plus edges on which the two digraphs thus obtained agree, denoted by $|\text{intersect}(d_i, d_j)|$. Note that we assume that both naming conflicts and abstraction conflicts have been resolved via reference to the enterprise ontology. We relax the identity requirement for nodes in relation to the owner attribute - two nodes are also deemed to be identical if they agree on the ID and nodetype and the owner attribute of at least one of the nodes is null. This permits us to deal with proto-models where some owner roles are yet to be assigned. The similarity measure is given by $\min(\{ |\text{intersect}(d_i, d_j)| / |d_i|, |\text{intersect}(d_i, d_j)| / |d_j|\}$, where for a digraph $d$, $|d|$ represents the total number nodes and edges in $d$. The threshold is a tunable parameter - setting it low would generate a large number of potentially incorrect identity relationships, while setting it too high might lead to potential identities being ignored by the tool. The similarity measure described above is one of several that could be used in this context, reflecting alternative intuitions and it is not our intention to suggest that this might be the best similarity measure to use. Much remains to be done in exploring the effectiveness of alternative means of assessing similarity and suggesting model identity.

When an identity relation is indicated between a pair of process models extracted from distinct source artefacts, it is useful to cross-validate these models, i.e., to determine if the models support each other. In our current implementation, we use model consistency as the basis for cross-validation. If a pair of process models deemed to represent the same process is mutually consistent, then they are viewed supporting each other. On the other hand, an inconsistent pair of models of the same process generates a trigger for analysts to manually check the corresponding source artefacts and also to manually resolve the inconsistency. We outline below a lightweight, structural approach to determining process model consistency that has been implemented in our tool. We note that a semantic approach to consistency would be more desirable, but is somewhat difficult due to the absence of consensus on the most effective means of describing BPMN semantics.

In the context of formal languages, two distinct theories in the language are deemed to be consistent if and only if a model (in the sense of model-theoretic semantics) exists that satisfies both theories. In our context, process models may be viewed as (syntactic) theories, or descriptions of processes, while individual process instances may be viewed as playing the role of semantic models (snapshots of the world being syntactically described). A pair of process models may therefore be deemed to be consistent if a process instance exists that satisfies both models. The consistency check that we have implemented performs lightweight, structural analysis of the digraphs obtained from BPMN models in the manner described above.

Let $m_1$ and $m_2$ be two graphical process models that we have initially determined to be identical. As with the similarity measure, we assume that naming and abstraction conflicts have been resolved with reference to an enterprise ontology. As before, we will permit pair of nodes to be deemed to be identical even if the owner role for one of them is undefined. We say that $m_1$ is consistent with $m_2$ (with $d_1$ and $d_2$ representing the corresponding digraphs, respectively) iff the following properties hold:

1. The sub-graphs within $d_1$ and $d_2$ defined by the nodes common to $d_1$ and $d_2$ are isomorphic.
2. For each incoming edge connecting a common node to a node that does not belong to the intersection in
Table 1. Figures 6 and 7 Correspondence

<table>
<thead>
<tr>
<th>Roles</th>
<th>Figure 6 (UML)</th>
<th>Figure 7 (Text)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portal</td>
<td></td>
<td>the system</td>
</tr>
<tr>
<td>Customer Service</td>
<td></td>
<td>the customer service representative</td>
</tr>
<tr>
<td>Customer</td>
<td></td>
<td>a customer</td>
</tr>
<tr>
<td>Nodes</td>
<td>Request process(Order)</td>
<td>submit an order for goods</td>
</tr>
<tr>
<td></td>
<td>order.CreditCheck=Approval</td>
<td>credit check passes</td>
</tr>
</tbody>
</table>

one digraph, there does not exist a corresponding incoming edge connecting the same common node in the other. Similarly, for each outgoing edge connecting a common node to a node that does not belong to the intersection in one digraph, there does not exist a corresponding outgoing edge connecting the same common node in the other.

Figure 7. BPMN Proto-Process Model Fragments Extracted from Text

Example:

Figure 7 ($m_T$) summarizes some fragments of a Sales Order proto-process model that has been extracted from a sample text using the R-BPD prototype. The (ontological) correspondences established in Table 1 between $m_U$ (Figure 6) and $m_T$ (Figure 7) provide an initial basis with which to determine consistency.

Application of the consistency check reveals the following:

- In $m_T$, the node and edge pair “credit check passes $\rightarrow$ ”

[Diagram]

Figure 8. Analyst Meditation and Inconsistency Resolution

, and the $m_U$ “CreditCheck = Approved $\rightarrow$ node and edge pair violate consistency rule (2).

- In $m_T$ the node and edge pair “[submits an order for goods $\rightarrow$ ”, and the $m_U$ “[Request process(Order)] $\rightarrow$ (Portal)” also violate consistency rules (2).

Such inconsistencies that arise during discovery may be either resolved in an automated or mediated manner. Given much of the available information has been extracted and summarized, such inconsistencies require only minimal analyst involvement and may even help unearth previously unknown and valuable process change information [21].

During the resolution of the first inconsistency (Figure 8 (a)), the analyst has chosen to differentiate the previously identified node and place it under control of ‘the system’ with the subsequent activity. Such information would in most cases be only available to an expert within the domain and may place the change out of scope of the R-BPDtk. Finally, the second inconsistency results in an update of the domain ontology signifying an association between “CD-Express”, “portal”, and “the system”. The acquisition of such information would be valuable during subsequent or revised iterations of discovery.
The $R$-BPD procedure continues until the correspondences and combinations of proto-process models have been established up to some fixed-point. We can then apply additional metrics to determine where further analyst mediation may be required.

5 Managing Traceability and Change during $R$-BPD

The extraction of process models from artefacts in an enterprise repository helps establish critical traceability links, which can be leveraged in a variety of ways. If a prototype model is deemed to be incorrect by an analyst and is found to have been extracted from a live artefact, an immediate alert (to the owners of the artefact, e.g., the authors of a document) is triggered. Such an alert does not oblige revision of the source artefact - it merely signals that disagreements (between the analyst in question and the artefact owners) on the process being described might exist and might require resolution. The actual resolution of such a disagreement is outside the scope of our tool. Similar alerts can also be issued to the owners of the source artefacts of process models that are changed for other extraneous reasons. Such alerts may be viewed as suggestions to revisit these source artefacts in light of the changes made to the process models extracted from them.

When an inconsistency arises, an analyst may either make a change to the model they know to be inconsistent, indicate that the models are in fact consistent, or conclude that the models actually describe different processes. To resolve inconsistency in the above example, the analyst may choose to change, remove or merge the conditional flow or subsequent activities. This will either highlight that the rapid extraction technique mis-interpreted the original artefact (mostly in text-model extraction), that the original artefact is inconsistent within the current situation, or that a previously unknown interpretation of the process actually exists. We apply the following techniques in R-BPD to handle inconsistencies.

6 Conclusion

In future work we propose to extend the toolkit to enable finer grained traceability between process models and source artefacts, relating elements of an extracted process model to components of source artefacts (such as paragraphs of text, or elements of a UML interaction diagram). This would permit a more sophisticated set of functionalities driven by analyst edits to generated proto-models. We also propose to explore alternative, and possibly semantic, notions of model identity and consistency. Finally, industry-scale empirical evaluation needs to be conducted.

References


