Extracting Data Objects and their States from Process Models

Andreas Meyer and Mathias Weske
Business Process Technology Group
Hasso Plattner Institute at the University of Potsdam
Potsdam, Germany
{Andreas.Meyer,Mathias.Weske}@hpi.uni-potsdam.de

Abstract—Business process model collections received much attention as option to store an organization’s process models and to make them available for further usage. Recently, also data objects and their states are studied intensively. However, process model collections often do not contain data objects. Generally, they focus on control flow structures of the process models. In this paper, we propose an approach to extract data objects from process models existing in a process model collection based on label analysis. In addition, the states of these data objects are extracted as well. After extraction, the data objects and their states are added to the corresponding process model. These resulting process models then can be used to, for instance, perform consistency checks between the process model and object life cycles in real world process model collections. First, the approach is presented generally for block structured process models independently of their notation before the approach gets adapted exemplarily to BPMN and EPCs to show the simplicity of adaptation to specific modeling notations. Finally, the approach is evaluated in a user experiment using the BPMAI model collection.

Keywords—BPM, Data Objects, Data States, Data Extraction Algorithm

I. INTRODUCTION

Business process management as “a systematic, structured approach to analyze, improve, control, and manage processes with the aim of improving the quality of products and services” [1] received much attention and big contribution steps in a variety of fields of applications especially during the last decade [2]. The first years focused on control flow, i.e., activities and their ordering, derivation, and execution. In recent years, further perspectives of business processes became important, most prominently the data perspective introducing and strengthening the concept of entities to be processed by activities. Over time, these entities have been called, for instance, business objects [3], business artifacts [4], or data objects [5], but generally they map to the very same concept on different views or levels of abstraction. From here, we refer to these entities as data objects.

Although the importance of data objects is well-known – for instance, the execution and automation of a business process requires detailed data object specification such that the process engine, executing the business process, knows how to proceed at all times – they are rarely modeled in process models. This statement holds true for existing publicly available or most organization internal process model collections. For instance, the SAP reference model [6] and the BIT process library from IBM [7] contain process models covering control flow only. The Signavio academic initiative (BPMAI) [8] provides some process models with explicitly modeled data objects, although the majority of the process models cover control flow only.

However, new techniques utilizing data objects and their states emerged recently. These techniques cover real world challenges as, for instance, consistency checking between process models and object life cycles [9]–[11] – object life cycles specify actions allowed to be performed on a certain data object – and the synthesis of object life cycles from process models [3], [12], [13] as well as data-aware process model abstraction [14]. Researchers developing these and similar techniques require process model collections, which contain process models with data objects and their states, for evaluation and refinement of the techniques before they can be applied in practice. Therefore, there exist two main needs for such process model collections: empirical research with respect to data-oriented techniques as well as their application in real world model collections.

Indeed, there exist real process models with data object annotations from organizations, which automate their processes with the help of information systems. But these process models are usually not shared with the research community such that we cannot utilize them for empirical research. Often, automation of processes is done manually, because access to data objects is not supported generically from process engines but needs to be developed by process experts and developers in contrast to control flow, which can easily be automated. Therefore, the underlying process models are mainly used for control flow and may lack data objects.

Usually, information about data is hidden in activity labels (if they are not anonymized as in the IBM collection). [15] describes that each activity label can be decomposed in up to three components: an action, a data object an action is performed upon, and a fragment providing further details (e.g., locations, resources or regulations). For instance, the activity labels ship products to customer and send invoice via email encode the information that data objects products and invoice respectively are processed in the corresponding activity. The actions performed are ship and send while the additional fragments provide insights about the additional resource involved and the regulation demanding a specific communication channel. Therefore, one solution to strengthen empirical research with respect to data related techniques is to enrich existing process model collections with explicitly modeled data objects and data states.
This paper introduces an approach consisting of a set of algorithms, which can be generally applied to most process models from various notations, because only activities, gateways, and control flow edges are used as input for data extraction. Further modeling constructs like events and message flow as well as gateways other than xor and and are not generally supported or only used rarely in business process modeling [16], [17]. Basically, for each source process model, i.e., the one, which shall be enriched with data objects and data states, a generic process model consisting of the named modeling constructs can be created independently from the source notation. The created – also generic – process model provides the researcher or stakeholder with explicitly modeled data objects and data states. Additionally, this approach can be adapted easily to be specifically tailored for a chosen notation, e.g., Business Process Model and Notation (BPMN) and event-driven process chains (EPC), utilizing all information provided by modeling constructs to improve the extraction quality.

The remainder of the paper is as follows. Section II introduces the fundamental notions including the generic process model followed by the extraction algorithms presented in Section III. First, the generic algorithms are introduced before they are adopted to BPMN [5] and EPC [18] process modeling notations. Afterwards, the generic algorithms get evaluated by enriching process models from the BPMN and EPC model collection and validating the results against a user experiment. Sections V and VI present related work and finally conclude the paper respectively.

II. PRELIMINARIES

This section defines the basic notions required to extract data objects and data states from control flow constructs to enrich a process model with this information afterwards. Consequently, a process model is the starting point for all algorithms and therefore, we define it first.

Definition 1 (Process model [19]): A process model is a tuple \( M = (A, G, D, R, C, F, \text{type}, L, \mu, \varphi) \), where \( A \) is a finite set of activities, \( G \) is a finite set of gateways, \( D \) is a finite set of data objects, \( R \) is a finite set of data states, \( C \subseteq (A \cup G) \times (A \cup G) \) is the control flow relation, \( F \subseteq (A \times (D \times R)) \cup ((D \times R) \times A) \) is the data flow relation, \( \text{type} : G \rightarrow \{ \text{xor, and} \} \) assigns to each gateway a type, \( L \) is a finite set of labels such that \( \tau \in L \) \((A, G, D, R, \text{and} L \text{are pairwaise disjoint})\), \( \mu : A \rightarrow L \) assigns to each activity a label, and \( \varphi : (G \times (A \cup G)) \rightarrow D \times R \) assigns conditions to control flow edges having an xor gateway as source.

Subscripts, e.g., \( A_M, D_M \), and \( \varphi_M \) denote the relation of the sets and functions to process model \( M \). Subscripts are omitted where the context is clear. The set \( A \cup G \) is referred to as nodes of process model \( M \). If \( \mu(x) \neq \tau, x \in A \), then \( x \) is observable in process model \( M \); otherwise \( x \) is silent. We call a gateway \( y \in G_M \) of \( M \) an xor (and) gateway, if \( \text{type}_M(g) = \text{xor} \) \((\text{and}) \). Additionally, an xor (and) gateway with two or more outgoing edges is called split (fork) and an xor (and) gateway with two or more incoming edges is called join (merge). Function \( \varphi \) specifies a condition comprising a data object with a corresponding data state for each outgoing control flow edge of a split. Further, each process model needs to satisfy a set of basic structural correctness criteria, which can be found in [19] along with the visualization and semantics of a process model with data annotations. In this paper, we expect a process model to be block structured and to be structurally sound, i.e., a process model has exactly one start and one end node, which are represented as silent activities, and each further node is on a path from the start to the end node.

The finally enriched process model aligns perfectly to Definition 1. The source process model, i.e., the one which shall be enriched with data annotations, only needs to contain a subset of the constructs introduced above. Already existing data information increases the quality of the result. But generally, the source process model is required to support the control flow constructs from above, i.e., nodes, control flow edges, activity labels, split conditions, and the functions \( \text{type} \) and \( \mu \). A split condition may not align with the format requested in the definition. Instead it may be plain text, which is structured similarly as activity labels are. Therefore, such split condition can easily be transformed into the required format using the techniques from the label analysis. The upcoming definition clarifies the notions of data object and data state as this is the information to be extracted from the activity labels of the source process model.

Definition 2 (Data object and data state): A data object is an entity or any piece of information being processed, manipulated, or worked with during business process execution. Data states represent the results of processing a data object in the process context. Thereby, each data state describes a specific situation of interest to the organization from the data object’s point of view.

A data object may represent, but is not limited to, documents, forms, database fields or tables, variables, messages, and products. In the healthcare domain, for instance, a data object may also represent a patient, because the doctor is examining her, i.e., the doctor works with or manipulates (in terms of medication) the patient. In the logistics domain, an activity send parcel processes a package (an entity) by sending it to the receiver. But a data object does not represent full IT systems or databases. A data state is closely related to a data object. A set of them, connected via a directed graph, shows the order of data states a data object may pass during process execution. Each of these data states is the result of an atomic activity or a set of activities; none is an intermediate result during execution of an atomic activity. An atomic activity describes an activity, which contains a single unit of work, which cannot be distributed amongst several resources. Next and last, we define an activity label, because each observable activity contains one and these build the basis for the extraction algorithms.

Definition 3 (Activity label): An activity label is an ordered list of words represented by a string describing an action, a data object an action is performed upon, and an optional fragment providing further details (e.g., locations, resources or regulations) [13].

With respect to empirical research from Mendling et al. [15], the majority of all activity labels conform to this definition. In the set of process models used by the authors, the SAP reference model [6], 94% of all activity labels contain at least an action and the corresponding object.
### III. Extraction

This section introduces how to automatically transform a process model without data object modeling into a process model with data objects and its data states explicitly defined as shown in Fig. 1. Thereby, the process models are expected to be on the operational level [20], i.e., the process model describes the relationships between the activities and their input and output requirements in terms of data objects with data states. The utilized process model describes a simple order and delivery process and aligns with Definition 1. The data objects shown in Fig. 1b have been extracted from the activity labels and the order of activities provided by the process model in Fig. 1a. The detailed extraction algorithms for generic process models are described below. Afterwards, the adaptation to specific process model notations is shown exemplarily for BPMN and EPCs.

#### A. Algorithms

The set of algorithms described here is aligned to the generic process model definition introduced above. Therefore, these algorithms can be applied to a large set of process modeling notations as most can be reduced to the set of concepts described in Definition 1 [21]. Further, we require four process modeling assumptions to hold to ensure convenient results: (i) outgoing arcs of xor splits are labeled with a condition as described in Definition 1, (ii) labeling is done homogeneously, i.e., the same object is always referenced by the same label, (iii) the object an action is performed upon as well as the action itself are present in each activity label although either of them may be missing in real world scenarios with respect to [22], and (iv) all activity labels follow the verb-object-style labeling [22]. We chose this labeling style as it is widely used and also widely accepted as a modeling style. An activity shall rather be labeled *analyze order* than *order analysis* to explicitly and unambiguously show the work comprised by the corresponding activity. Further, many activity labels violating assumption (iv) can be transformed into the appropriate style using the techniques introduced from Leopold et al. [23]. The authors provide means to identify the used labeling style and to transform most of them into a verb-object-style for English language activity labels. Subsequently, these techniques could be performed as preprocessing step to the extraction process, which comprises the following main steps:

1. label analysis to determine the action and the object (see Algorithm 1)
2. specify data output of activities (see Algorithm 2)
3. specify data input of activities (see Algorithm 3)

The actions performed within the first step, the label analysis, are shown in Algorithm 1. For each activity of the process model, the label is parsed and each word gets tagged with respect to its grammatical function as for instance verb, noun, adjective, or preposition (see lines 1 to 3). Then, the additional information, which is represented as prepositional phrase if existing, gets removed from all activity labels because it has no influence on data object and data state retrieval (see lines 4 to 6).

Afterwards, the tagging is verified. If the verification fails, it needs to be adjusted either manually or automatically. As the failure rate is comparably low, manual correction would be feasible. However, usually, similar labels are affected by mis-tagging in different process models, e.g., *ship products* because *ship* is tagged as noun although used as verb. Additionally, the labeling style assumption strongly limits the potential structures of an activity label. Therefore, we provide automatic means to correct the unverified tags. The first word of the activity label is a verb with respect to the chosen verb-object-style of labeling and thus, needs to be tagged that way. If it is not tagged as verb, the tag is changed accordingly (see lines 8 to 10). Next, the verification checks for existence of a noun tag. A fail results in tagging all words except the first one, which might be classified as noun, accordingly (see lines 11 to 13).

Consider activity *Ship products to customer* in Fig. 1, the activity label after tagging is *Ship_VB products_NNS* stating that *ship* is a verb in third person singular and *products* is a noun in plural form. The fragment containing the resource information where to ship the products has been removed during this step.

<table>
<thead>
<tr>
<th>Algorithm 1: Label analysis for data object and corresponding action recognition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: for all activities do</td>
</tr>
<tr>
<td>2: tag the words of the label regarding their grammatical function by parsing them with the Stanford parser</td>
</tr>
<tr>
<td>3: end for</td>
</tr>
<tr>
<td>4: for all tagged activity labels do</td>
</tr>
<tr>
<td>5: remove prepositional phrase from tagged activity label</td>
</tr>
<tr>
<td>6: end for</td>
</tr>
<tr>
<td>7: for all tagged activity labels do</td>
</tr>
<tr>
<td>8: if first word of activity label is not tagged as verb then</td>
</tr>
<tr>
<td>9: tag first word as verb</td>
</tr>
<tr>
<td>10: end if</td>
</tr>
<tr>
<td>11: if no word of activity label is tagged as noun then</td>
</tr>
<tr>
<td>12: tag all appropriate words except the first one as noun, i.e., all words potentially tagged as noun</td>
</tr>
<tr>
<td>13: end if</td>
</tr>
<tr>
<td>14: end for</td>
</tr>
</tbody>
</table>

After analyzing the labels, tagging the words towards its grammatical function, and ensuring existence of a verb and a noun, the input and output data objects and corresponding states need to be identified from this information and added to the process model. Adding output data objects is of comparably low complexity as Algorithm 2 shows. In contrast, determining the input data objects is a rather complex task, which relies on heuristics described below and summarized in Algorithm 3. Both algorithms analyze the process model towards patterns, which target the structure of the model in terms of precedence and successorship relations between modeling constructs. Thereby, a pattern comprises two nodes – at least one being an observable activity – and the ordering relation between them. Such patterns are, for instance, an activity precedes another activity, it succeeds an and fork, or it precedes an end node. A complete set of patterns comprises all combinations of precedence and successorship of an observable activity with respect to an observable activity, an xor (and) gateway, or a silent activity (precedence relation for
an end node and successorship relation for a start node). Each pattern can unambiguously be grouped towards its influence on the input or output data objects of the activity (from here, activity implicates observable activity) comprised by the pattern. Precedence relations influence the data output specification while successorship relations influence the data input specification.

Next, we explain the algorithms to specify the output and input data objects. Both algorithms are applied to each activity of a process model and based on the pattern determined for this activity, the corresponding data objects and data states are extracted. First, all output data objects are specified. Secondly, the input data objects are specified. This order of algorithm application is essential for the extraction results, because the latter utilizes the output information within its heuristics for the input specification. Thereby, both algorithms follow four assumptions: (i) an activity preceding a split or fork influences the activities within the block, (ii) an activity succeeding a join or merge is influenced from this block, (iii) each activity requires at least one output data object, and (iv) each activity, which is not directly succeeding the start node, requires at least one input data object.

Data output specification from Algorithm 2 first checks for the currently selected activity whether it precedes an xor split. If the check evaluates to true, a data object with the corresponding data state is retrieved from each condition of the outgoing edges of the xor split (see lines 1 to 4). Otherwise, i.e., the activity does not precede an xor split, the required information for the data object is extracted from the activity label by two functions. Based on the tagging done in Algorithm 1, the first function takes the noun of the activity label as object and the second one transforms the verb in past participle form and takes this result extended with the adverb if existing as the corresponding data state (see line 6). Finally, all data objects, retrieved on either way, are associated to the activity as output data objects considering only not yet associated data objects utilizing label and data state matching to avoid duplicates (see line 9). Then, the algorithm starts over again with the next activity until each got assigned the appropriate output data objects.

Based on the tagged activity label Ship_VB products_NNS, the output data object Product in data state shipped is extracted and annotated to the process model accordingly. The data state shipped results from the past participle form of the verb while the noun is directly taken as data object.

Having succeeded with steps one and two, the specification of input data objects remains open. The procedure for each activity is described in Algorithm 3. Comparably to the output specification, the patterns are matched against the local process structure. In contrast to the output specification, each pattern requires a different handling to identify the input data objects of the activity. Additionally, this input identification also requires information about output data objects of various activities preceding the currently selected one. Therefore, complexity and length of the algorithm is increased. For space reasons, the handling for two patterns is outsourced to Algorithms 4 and 5 respectively as noted in lines 21 to 22 and lines 26 to 27.

Lines 1 and 2 indicate that an activity, which succeeds the start node, does not get assigned input data objects. In case the currently selected activity succeeds an other activity (see line 3), all output data objects of the selected one are retrieved and all paths from the start node to the selected activity are identified. For each of these data objects, we analyze each path, whether the data object has been used as output to an activity on that path. If so, we create a new data object with the same label and assign to it the data state of the output data object associated to the activity with the smallest distance to the currently selected activity, i.e., from the set of activities
Algorithm 3: Data input specification

1: if activity succeeds start node then
2: //no input data object for activity
3: else if activity succeeds other activity then
4: for all output data objects of activity do
5: for all paths leading to activity do
6: if data object is used before as output then
7: object = object(outputDataObject);
8: dataState = dataState(lastUsed(object));
9: new dataObject = <object,dataState>;
10: end if
11: end for
12: end if
13: if #dataObject == 0 then
14: for all output data objects of other activity do
15: new dataObject = <object(outputDataObject),
  dataState(outputDataObject)>;
16: end for
17: end if
18: else if activity succeeds xor split then
19: new dataObject = <object(condition),dataState(condition)>;
20: end if
21: else if activity succeeds xor join then
22: //see Algorithm 4
23: else if activity succeeds and fork then
24: //depends on predecessor of fork and utilizes the
  corresponding computations shown in this algorithm
25: else if activity succeeds and merge then
26: //see Algorithm 5
27: activitySucceedsAndMerge();
28: end if
29: for all dataObject do
30: activity.addInputDataObject(dataObject);
31: end for

Algorithm 4: Activity succeeds xor join

1: for all branches b of xor block do
2: for all output data objects of activity do
3: if data object is used as output in b then
4: object = object(outputDataObject);
5: dataState = dataState(lastUsed(object));
6: new dataObject = <object,dataState>;
7: else if data object is used in condition of b then
8: new dataObject = <object(condition),dataState(condition)>;
9: end if
10: end for
11: if #dataObject == 0 then
12: for all output data objects of first preceding
  activity of join do
13: new dataObject = <object(outputDataObject),
  dataState(outputDataObject)>;
14: end for
15: end if
16: end for

belonging to this path and having the data object as output without the currently selected activity, the data state is retrieved from the activity to be executed last on this path (see lines 4 to 12). If none of the output data objects of the currently selected activity is output to some other activity on some path from the start node to the current selected activity, for each output data object of the activity directly preceding the currently selected one, a new data object is created with the same label and data state (see lines 13 to 17).

Lines 18 and 19 handle the appropriate xor split pattern. Here, the conditions on the outgoing edges of the xor split are considered for data object creation. Each condition maps to one data object with a corresponding data state. Thereby, the functions already introduced in Algorithm 2 are used.

As mentioned above, the pattern describing the input data object handling if the activity succeeds an xor join is described in detail in Algorithm 4. Thereby, we assume that the xor block influences this activity, so that it provides the input data objects. First, all branches of the xor block closed by this xor join are identified and the output data objects of the succeeding activity are retrieved. For each branch and data object, we check whether the data object is output to some activity of this branch. If so, a new data object with the same label and the data state belonging to the activity with the smallest distance to the currently selected one is created (see lines 3 to 6 and compare with lines 4 to 12 of Algorithm 3). Otherwise, we check the condition of the branch for data object matching. If the evaluation returns true, the new data object is retrieved from the condition (see lines 7 to 9 and compare with line 3 of Algorithm 2). If none of the output data objects is used as additional output in a specific branch nor does the corresponding condition map to any of the output data objects of the currently selected activity, for each output data object of the activity first preceding the xor join on the specific branch, a new data object is created with the same label and data state (see lines 11 to 15). These steps are repeated for each branch.

The detailed handling with respect to the pattern describing that the activity succeeds an and fork is omitted in the algorithm representation, because the handling depends on the direct predecessor of the fork (see lines 23 and 24 in Algorithm 3). Therefore, we consider the predecessor of the fork and the currently selected activity for pattern matching while applying Algorithm 3.

If the currently selected activity succeeds an and merge, similar steps need to be undertaken as for the xor join successorship. Details are shown in Algorithm 5. Again, we assume that the and block influences this activity, so that it provides the input data objects. First, all branches of the and block closed by the merge are identified and the output data objects of the currently selected activity are retrieved. For each branch and data object, we check whether the data object is output to some activity of this branch. If so, a new data object with the same label and the data state belonging to the activity with the smallest distance to the currently selected one is created (see lines 3 to 6 and compare with lines 4 to 12 of Algorithm 3). If none of the output data objects of the currently selected activity is – differently to Algorithm 4 – output to some other activity in any branch in the and block, for each output data object of all activities first preceding the and merge on some branch, a new data object is created with the same label and data state (see lines 10 to 16).
Algorithm 5: Activity succeeds and merge

1: for all branches b of and block do
2:   for all output data objects of activity do
3:     if data object is used as output in b then
4:       object = object(outputDataObject);
5:       dataState = dataState(lastUsed(object));
6:       new dataObject = <object, dataState>;;
7:     end if
8:   end for
9: end for
10: if #dataObject == 0 then
11:   for all activities first preceding and merge do
12:     for all output data objects of activity do
13:       new dataObject = <object(outputDataObject), dataState(outputDataObject)>;
14:     end for
15: end for
16: end if

After identifying the input data objects of the currently selected activity based on the corresponding pattern, these are associated to the activity as input data objects (see line 30 in Algorithm 3). Thereby, only not yet associated data objects are considered utilizing label and data state matching to avoid duplicates. Afterwards, Algorithm 3 is applied to the next activity of the process model.

Referring again to activity Ship products to customer, the input data objects are extracted by Algorithm 4, which is basically part of Algorithm 3, because the mentioned activity directly succeeds an xor join. The extracted input data objects are Products in data state in stock and Products in data state manufactured. As the only output data object of activity Ship products to customer is Products, the activities of the branches of the xor block are checked accordingly. The first mentioned input data object results from the condition of the control flow edge leading from the xor split to the join. Data object Products is used as output to at least one activity of the other branch of the xor block, namely Manufacture product. As this is the only usage, it is also the last usage of that data object. Therefore, the second input data object is the output of this activity, i.e., Products in data state manufactured.

Finally, after successfully performing the three steps for data object extraction, duplicated data objects may be combined to improve clarity and readability of the process model. Duplicates appear, if the same data object is used as input or output to different activities. These can easily be consolidated to associations with one object in the process model. Following, the total number of used modeling elements gets reduced. In Fig. 1b, this reduction has been applied as the reader can determine by, for instance, looking on the first two activities and their associations to the same data object (Order in data state analyzed).

The process model may also contain data objects prior applying the algorithms introduced above. In these cases, the data objects get preserved and additionally extended by the newly extracted ones. Duplicate detection within the applied algorithms also considers initially existing data objects.

B. Adaptation to existing Notations

The algorithms above can be extended to also incorporate further modeling constructs of various process modeling notations. Therefore, the complete approach presented in this paper can easily be adapted to a process modeling notation of choice to enrich process models with data objects in there. Alternatively, a process modeling notation of choice can be mapped to the above algorithms such that each modeling construct is linked to one introduced in Definition 1. This subsection is dedicated to present the alignment for the process modeling notation BPMN followed by an algorithm extension for EPCs.

As BPMN is a very powerful notation with a large number of modeling constructs, we restrict the alignment to those constructs frequently used [16], [17]. In detail, the alignments are the following: (i) BPMN activities are mapped to observable activities, (ii) BPMN start and end events are mapped to silent activities, (iii) BPMN xor (and) gates are mapped to xor (and) gates, (iv) existing BPMN data objects are mapped to data objects including input and output associations to activities and existing data states, (v) receiving (sending) BPMN messages are mapped to input (output) data objects of the corresponding activity, (vi) preceding (succeeding) receive (send) BPMN message events of an activity are mapped to input (output) data objects, and (vii) BPMN pools and lanes are ignored as these do not influence data object specification utilizing above algorithms. After applying this mapping to a BPMN process model, the proposed approach can be run as is. Thereby, only data objects and associations are added, which have not been existing before. For data objects without data state specification existing before, the missing data states can be retrieved from the activity label, if it encodes the corresponding data object. Otherwise, the user has to decide about the data object after the enrichment computation. She may decide to add the data state information manually, to keep the data object as is, or to remove it from the process model.

Considering all constructs of BPMN leads to issues especially with respect to the inclusive or and the complex gateways. The internal behavior of the complex gateway can hardly be mapped to an xor or an and gateway because of its unpredictive behavior specification within different process models. Similarly, the inclusive or gateway is difficult to map as it functions as m-out-of-n discriminator with n being fixed by the number of branches of the inclusive or block. But m varies even between process instances such that the behavior is also unpredictable in many cases. However, sometimes the inclusive or block can be replaced by a number of other common (see above) modeling constructs [24].

EPCs provide further modeling constructs, which increase the information input for data object extraction. Therefore, it is valuable to extend the algorithms from above instead of aligning the modeling constructs. Explicitly, the according modeling construct is the event, which specifies the income and the outcome of an activity respectively depending on whether it precedes or succeeds the activity – called function within EPCs. The extension comprises event consideration for data object specification and a removal of start and end node consideration as these are covered by events. Events preceding the currently selected activity are of interest for the input data objects while events succeeding the activity are of interest for the output
data objects. Therefore, Algorithm 1 needs to be extended with means to analyze event labels as well. For them, the same assumptions as for activity labels hold except that the labeling style follows the widely used technique of an object followed by an auxiliary verb and the action, e.g., products are shipped. Subsequently, the set of analyzed modeling constructs need to be enhanced to activity and event labels as first step for the overall approach. Then, the two other algorithms need to be extended with if statements applying to the corresponding patterns activity succeeds event and activity precedes event. Based on above mentioned matching schema, the data object and the corresponding data state are retrieved from the event in addition to the activity label extraction. The information extracted from activities and events gets combined such that each extracted data object is associated to the corresponding activity ignoring duplicates.

IV. Evaluation

We conducted an experiment with 29 participants to evaluate the appropriateness and usefulness of the extraction approach introduced in this paper. The participants of the user experiment are students and researchers in computer science. None of them has been involved in the development of the algorithms or their implementation. The level of experience ranges from beginner to expert; about half the participants have a strong background in business process management. The questionnaire\(^1\) was separated in three main parts tackling the extraction approach. The first one dealt with annotating given process models with data objects while the second one asked the participants to rate the data annotations for specific activities induced through application of Algorithms 1 to 3. The third part asked for the appropriateness of input and output specifications as well as the understandability for entire process models. With respect to Section I, these scenarios comprise the chance to run further analyses using the enriched process model or to further refine the result towards an executable process model. The utilized process models are taken from the BPMAI model collection [8] to utilize real process models from different users describing several scenarios.

We used ten different process models for the survey – reusing some in different parts – with three to five process models in each part. Each process model was presented as BPMN diagram [5] without further textual descriptions, i.e., the participants based their decisions on the presented graph only analogously to the extraction algorithms. Complexity-wise, the process models contained at average ten observable activities, four gateways, and two silent activities summing up to 16 nodes at average per process model with a standard deviation of 3.6 nodes and a median of 15.5 nodes, i.e., a common complexity often found in process model collections. In the experiment, the participants have seen the corresponding process models for the first time such that they cannot utilize previous knowledge to allow comparability of the results. All surveys were identically structured and utilized the same process models.

Below, we will present the results of the experiment, which had the goal to tackle the following aspects:

- comparison of automatic and manual data annotations
- quality and usability of data annotations in different usage scenarios
- additional insights to and knowledge about the business process represented by the process model
- understandability of a resulting process model

The comparison results between automatic and manual data annotations are provided in Fig. 2 (part one of the questionnaire). The bars show the coverage in percent of the number of data objects (black bars) or data states (dark gray bars) similarly identified by the algorithms and humans. The light gray bars indicate the coverage of activities completely specified with data objects and their states. This conformity is shown separately for input and output data objects of activities as well as combined for both types of data objects. Thereby, we distinguish an exact label match (group one, three, and five) and a synonym match, because humans may consider external knowledge and personal taste for annotation such that, for instance, a bill data object is extracted from a send invoice label or an order is changed to purchase order or vice versa. Humans also tend to add extensions like report for assessment activities. With synonym match, we consider data objects or states providing the same meaning and understanding although string matching might fail.

For all criteria, the data object extraction (black bars) reaches the highest conformity. The difference between data object and state extraction conformity is quite close for output data objects, but quite large for the input data objects due to the heuristics applied here. The lowest values are generally reached by the combination of data object and data state extraction (light gray bars) for natural reasons, because it represents the intersection of the other two criteria.

Altogether, the annotation done by our approach is generally quite close to the humans annotations with the majority of

\(^1\)http://www.surveymonkey.com/s/dataExtraction
Fig. 3. Rating of data annotation quality and usefulness

data objects and states being compatible to each other (see Fig. 2a). Thereby, output specifications provide closer results due to the heuristics used for input specification. However, the main problems for both annotation procedures are abstract or confusing activity labels, i.e., too abstract or detailed abstraction level as well as syntax and semantic errors. Evidence shows that the quality of activity labels plays a major role with respect to the results to be expected from our approach – and from the humans. A good label quality refers to labels, which actually describe the work performed by the corresponding activity with clear and concrete statements and align to the structure of the process model. This also includes an appropriate level of abstraction, which shall be comparable throughout the complete process model. For instance, activity label send invoice by email consists of a good quality, because it states clearly what happens without too many or too few details. In contrast, labels of poor quality either abstract completely from the work to be performed in the activity, contain ambiguous statements, or are very detailed by, for instance, making proposals about what should be done within activity execution; e.g., activity label consider other causes of distress and pain is very specific and also humans have difficulties to extract data objects, which might be affected by such activity. Following, clear and good structured activity labels also increase the conformity between automatic and manual extraction of data objects.

Referring to our user experiment, removing process models, which majority of activities is poorly labeled, the results increase by 25 percentage points at average (see Fig. 2b). For example, the association of data objects and states on a synonym basis increases from 37% to 61%. The remaining difference reflects the different thinking of humans as well as their usage of external knowledge.

Fig. 3 visualizes the results from part three of the questionnaire: the assessment of the quality of the automatically added data annotations on a Likert scale from 1 (excellent) to 6 (not usable). The participants rated the data input and output specifications of complete process models separately as well as combined (bar groups one to three) and the usability (for further analysis or as starting point for refinement towards executable process models), the ability to derive new insights from the process model, and the level of confusion triggered by data objects. Especially for models 2 to 5, the figure shows very convenient results and confirms the good quality of the automatic data annotations. Model 1 contains almost only abstract activity labels from the healthcare domain such that the extracted information is as indicated above of low use for process understanding or utilization.

Including all models, the arithmetic averages are 2.6 ± 0.8, 2.7 ± 0.7, and 2.7 ± 0.8 for the data specifications and 2.6 ± 0.6, 2.8 ± 0.8, and 3.0 ± 1.2 for usability, insights, and the level of confusion respectively. Ignoring model 1, the averages decrease to 2.2 ± 0.3, 2.4 ± 0.4, and 2.4 ± 0.4 for the data specifications and 2.4 ± 0.3 for usability and insights and 2.4 ± 0.4 for the level of confusion, which means stable and high-level results for different process models with a good label quality from different domains. Additionally, we identified a direct correlation between all six evaluated aspects, which again correlate to the quality of the activity labels as mentioned above and the structure of the process model. For instance, long sequences with many different data objects often result in repetition of the activity label in the data object and therefore, complexity is increased without adding value. In contrast, process models with several building blocks benefit from the approach and provide insights about data usage and manipulation throughout the complete process model.

In part two of the questionnaire, the participants were asked to rate the appropriateness of the data annotations for single activities while knowing the complete annotated process model.

Most annotation results have been either rated very low (worse than score 4 at average) or very high (better than score 2 at average); only few scores are around three. The overall scores are 2.2 ± 1.1 for input data objects and 2.3 ± 0.6 for output data objects. These results demonstrate the importance of the label quality. Ignoring activities with a poor label quality decreases these numbers to 1.6 ± 0.3 and 2.1 ± 0.6 respectively leading to an appropriateness rating between good and excellent and shows the general applicability for well labeled activities.

Summarizing the insights from the user experiment, the results are very promising but highly depend on the label quality. Reflecting on the results for process models mainly consisting of activities with good quality labels, the comparison of automatic and manual data annotation has a conformity of about two thirds with respect to the number of data objects and data states. The resulting automatically annotated activities and process models achieved high ratings with respect to annotation quality, usability in different scenarios, knowledge gaining, and understandability.

Good label quality is achieved by utilizing modeling guidelines in conjunction with a taxonomy or glossary specifying the correct term for each action and object [25], [26]. Generally, label quality is a requirement of the modeling process and can be enforced by the modeling tool.

The experiment showed that the resulting process models can be directly used for empirical research or as basis for process automation, if the labels are clear, concrete, and aligned with the process structure. In contrast, ambiguous, very detailed, or very generic labels lead to process models, which may act as starting point to annotate the process model with data objects and their states by providing insights about the manipulations performed by the activities. A proof of concept implementation, used as basis to evaluate the algorithms presented in this paper, is available at http://bpt.hpi.uni-potsdam.de/Public/ExtDO.

V. RELATED WORK

The approach described in this paper relies on findings from [15], where the authors determined a general label construction schema consisting of three building blocks. The insight of existence of an object and an action manipulating this object is
the basis for our approach as we require both building blocks to specify the data objects being input or output to an activity. Additionally, further activity label analysis has been performed. Leopold and colleagues analyzed the labels with respect to its grammatical structure and determined seven different labeling styles [22], [23]. From these, they chose the verb object style labeling as the reference labeling style, because this is the one to be recognized as good practice for modeling processes. Therefore, the authors provide means to transform a given activity label of non-regular-style into verb-object-style [23]. These transformations can be used as pre-step to our approach to increase the number of process models, which can be enriched with data objects and data states.

To be able to gather information from activity labels, they need to be analyzed with natural language processing techniques. There exist several frameworks to do so. Two of the well-known ones are the Stanford part of speech tagger [27] and the KPMML system [28]. The label analysis results in words tagged towards their grammatical function, e.g., verb in infinitive form. We use this output to reason about the data objects and their states. In our implementation, we use the Stanford tagger with respect to the easy integration as jar library.

Another stream of research deals with data in process models. For instance, there are approaches to derive object life cycles from a process model [3], [12], [13], which also need to determine data objects and the corresponding data states in the process model. However, the existing approaches in this regard require process models being annotated with data. Thus, these approaches can be extended with the extraction approach introduced in this paper to derive object life cycles from process models consisting of control flow only.

Data in business processes leads to object-centric modeling approaches [4], [29], where a process is modeled by the involved objects with each having a life cycle a multiple objects synchronize via their state changes. In contrast, in activity-centric modeling, data objects are used as second class modeling construct while activities describe the process behavior. Here, data objects describe pre- and post-conditions to activities describing under which conditions can be enabled and what is the output produced by the activity [30]. The approach presented in this paper focuses on enriching an activity-centric process model with data objects rather than creating an artifact-centric process model. Modeling methodologies for object-centric processes are presented in literature [4], [31]. Additionally, an activity-centric process model with annotated data objects can be transformed into an object-centric process model [12].

VI. CONCLUSION

We presented an approach to extract data objects and the corresponding data states from the activity labels of a process model. The approach consists of three steps including the analysis of the activity labels and the extraction and specification of the output data objects followed by the input data objects. The resulting process models can be used, amongst others, for analyses concerning data or the derivation of object life cycles. The proposed extraction approach is based on the need of process models with annotated data objects for the mentioned techniques although most process models in currently existing process model collections contain only control flow descriptions.

A tool implementing the algorithms described in this paper would provide benefits to researchers working in the field of data-aware business process management with need to evaluate their algorithms and approaches as well as to companies moving from documentation to execution with their processes. The latter receive comprehensive insights about the usage of data objects throughout their processes. Indeed, for single processes the enrichment may be done manually, but considering a large process collection with hundreds of models requires much time for manual processing. Here, such a tool would highly support and decrease effort to be spent for data modeling.

In future work, we plan to use the introduced approach to enrich process models with data objects and data states to utilize these for evaluation of data abstraction and data consistency approaches.

REFERENCES


