From product data model to process model

Kamphuis, J.; Vanderfeesten, I.T.P.; Reijers, H.A.; Hattem, van, M.

Published: 01/01/2008

Document Version
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the author’s version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 02. Jan. 2019
Abstract. Product based workflow design (PBWD) is an upcoming practice in the field of business process redesign. PBWD takes the product structure as a starting point. A product data model (PDM) is such a product description. For the translation from a PDM to a process model we have distinguished two different, yet coherent perspectives: the user-perspective and the designer-perspective. These two perspectives form a classification framework for the algorithms that perform such a translation from a PDM to a process model. Based on this framework we have derived seven such algorithms. A description of the algorithms and their characteristics is provided in this paper.

Table of contents

1 Introduction ..................................................2
2 Strategies on the PDM ....................................3
  2.1 Construction ...........................................3
  2.2 Process execution ....................................4
  2.3 Coherence .............................................4
  2.4 Framework ............................................5
3 The algorithms .............................................6
  3.1 from PDM to PM, algorithm Alpha ..................6
  3.2 from PDM to PM, algorithm Bravo ..................9
  3.3 from PDM to PM, algorithm Charlie .................12
  3.4 from PDM to PM, algorithm Delta ..................14
  3.5 from PDM to PM, algorithm Echo ....................17
  3.6 from PDM to PM, algorithm Foxtrot .................20
  3.7 from PDM to PM, algorithm Golf ....................23
4 Related work ..............................................26
5 Conclusions and future work ..........................26
References ..................................................27
Appendix A: Benchmark Set ..............................28
1 Introduction

In the field of business process redesign an evolutionary and a revolutionary approach can be distinguished. Product based workflow design (PBWD) is a revolutionary approach [1, 4]. PBWD takes the product specification as a starting point in order to derive a favorable new design of the workflow process [10].

A product data model (or PDM for short, introduced in [2]) is a static description of an administrative, information based product that is produced in a business process (e.g. a mortgage loan, an insurance claim, a social benefit). The PDM is similar to a Bill-of-Material (BoM) from the manufacturing area, but while a BoM represents a physical product, the PDM represents an informational product. Both PDM and BoM represent the step-by-step structure of the product [1, 4, 10].

PBWD is mainly applied manually to this day, although there are initiatives to automate this process [3]. This paper describes and classifies seven algorithms that can be used for deriving workflow process models based on the PDM, as suggested in [1]. The variety in algorithms and their classification is an important step to make the automated PBWD a usable practice.
2 Strategies on the PDM

To understand more of how the business process can be supported based on the PDM, we look at the different strategies that can be used to generate a process model from a PDM.

We distinguish two perspectives to identify the different strategies:

- The construction perspective: the translation is considered from a point-of-view on the way a process model can be constructed from the PDM.
- The execution perspective: the translation is considered from a point-of-view on the behavior of the generated process model while being executed.

The criteria listed in sections 2.1 and 2.2 are selected to indicate the differences between our algorithms. In our opinion these algorithms form a usable set of translations. However, the set of criteria is expandable. With new or other criteria, new algorithms may evolve that result in process models with different characteristics. It is up to the end user and the designer to determine the relevance of the criteria for a specific situation.

2.1 Construction

In the construction perspective we look at how the process model can be constructed given the PDM. It is the perspective of the designer or the process modeler.

For the construction we identified three criteria:

1. The representation of the process model. There are many different representations of process models available. Right now, our algorithms will produce either a Petri net [8] (algorithms Alpha, Bravo, Charlie, Echo, Foxtrot and Golf) or a YAWL model [7] (algorithm Delta).

2. The order in which the algorithm translates the PDM to the process model. This can be done in the following manners:
   - Top-down: starting with the root element of the PDM and working towards the leaf elements, following dependencies as set in that PDM.
   - Bottom-up: starting with the leaf elements and working towards the root element step-by-step, based on the enabling of possible next steps.
   - Mix of top-down & bottom-up: the PDM will be examined in both the top-down and the bottom-up manner as described above, in order to combine the best of both manners.
   - Middle-out: every data element and every operation is translated in arbitrary order. So there is no specific order in which the translations of data elements and operations are added to the process model. Next, these translated parts are connected to each other in the right manner, respecting the PDM structure.

3. The focus of the translation. The process model can be generated from the PDM with the focus on either the operations or the data elements. With a different focus, different process models will be produced.
2.2 Process execution

The process execution perspective looks at the behavior of the process model derived from a PDM when it is being executed. This behavior is a direct result of the structure of the process model. This is the typical perspective that is relevant to the user.

We have identified three important criteria that influence the execution behavior:
1. The moment of choice in the process model. During execution, at some point in time it has to be decided which alternative (if any) will be used to construct a data element. It is possible that before doing anything else, all choices need to be made. Or it might be useful to defer choices as long as possible. The structure of the process model determines the moment when such a choice has to be made.

2. The eagerness of the execution. The structure of the process model specifies how much data can be determined.

   - **Strict**: it is possible to construct process models in which, upon execution, only those data elements will be determined that contribute to the determination of the root element.
   - **Overcomplete, but restricted**: upon execution more data elements than needed to construct the root element might be calculated. However, once the root element is determined, execution ends.
   - **Overcomplete and unrestricted**: even after the root element is determined, execution of the process model might continue.

3. Concurrent execution of steps. The possibility of several activities or steps to be executed simultaneously. For Petri nets this means that at least two transitions are enabled at the same time, and execution of one of those transitions does not change the other being enabled.

2.3 Coherence

As the concerns of the process modeler and the user can not be strictly separated, the construction and the process execution perspective influence each other.

The end user will usually have certain wishes or requirements to which the process model has to conform. These user-criteria restrict the amount of freedom the process modeler has. For instance, if the end user wants the moment of choice to be late, the eagerness to be strict and the concurrent execution of steps to be possible, than with the current set of algorithms the process modeler can only build the process model in a middle-out order.

Vice versa shall the choices of the process modeler affect the model the end user will have. For instance, the representation of the process model (Petri net or YAWL), chosen by the process modeler, will obviously affect the appearance of the model.

The criteria within the process execution perspective influence each other as well. For instance, if the user wants the moment of choice to be early, than it is logical that the eagerness will be strict; if in an early stage is chosen which data elements are needed, then it makes sense to only produce these chosen elements, instead of producing more than needed.

It may be clear that the criteria within the construction perspective also influence each other. E.g. if a YAWL-representation is chosen, than with the current set of algorithms the process model can only be constructed in a middle-out order.

Summarizing the above, the algorithms for the translation from PDM to process model will depend on both the criteria from the construction perspective and the criteria from the process execution perspective.
### 2.4 Framework

The information above can be summarized in the following matrix. This also provides us with a framework to classify our algorithms, as is also shown in the matrix. An elaborate description of the algorithms is given in the next chapter.

As claimed earlier in chapter 2, the set of criteria is expandable. This is also true for the possibilities of some criteria, which is visualized with the dashed lines in the framework.

<table>
<thead>
<tr>
<th>Strategy perspective</th>
<th>Criterium</th>
<th>Possibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Representation</td>
<td>Petri net</td>
</tr>
<tr>
<td>Construction</td>
<td>Order</td>
<td>Top-down</td>
</tr>
<tr>
<td></td>
<td>Focus</td>
<td>Operations</td>
</tr>
<tr>
<td></td>
<td>Moment of choice</td>
<td>Early</td>
</tr>
<tr>
<td>Process execution</td>
<td>Easiness</td>
<td>Strict</td>
</tr>
<tr>
<td></td>
<td>Concurrent execution of steps</td>
<td>Possible</td>
</tr>
</tbody>
</table>

- Framework visualization with algorithm details:
  - From PDM to PM, algorithm Delta
  - From PDM to PM, algorithm Charlie
  - From PDM to PM, algorithm Bravo
  - From PDM to PM, algorithm Alpha
  - From PDM to PM, algorithm Echo
  - From PDM to PM, algorithm Golf
  - From PDM to PM, algorithm Foxtrot
3 The algorithms

This section describes a set of algorithms to derive a process model from a PDM. For each algorithm a classification based on the two perspectives described in the previous section is given. Next, the algorithm is clarified with a step-by-step description and a small example. The same PDM (see figure) is chosen as an example for all algorithms. This makes it easier to compare the algorithms with each other. The example can also be found in Appendix A under the name of "test04 ABC2".
Then, possible shortcomings of the algorithm and their solution are explained and the characteristics of each model are elaborated.

3.1 from PDM to PM, algorithm Alpha

Classification:

<table>
<thead>
<tr>
<th>Representation</th>
<th>Petri nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order</td>
<td>Top-down</td>
</tr>
<tr>
<td>Focus</td>
<td>Data elements</td>
</tr>
<tr>
<td>Moment of choice</td>
<td>Early</td>
</tr>
<tr>
<td>Eagerness</td>
<td>Strict</td>
</tr>
<tr>
<td>Concurrent execution of steps</td>
<td>Possible</td>
</tr>
</tbody>
</table>

Description of algorithm:

Summary: All alternative paths are elaborated top-down from the root element. Starting from a process model that only contains an input place, a transition for the root element and an output place, the final process model is obtained by replacing each transition with a subnet adding more detail.

Step-by-step: The algorithm starts with the root element ‘X’ (in the example this is data element ‘A’). A transition X is created for this data element. Also, an input and an output place are created and connected to transition X (STEP 1).
Then, if data element X is a leaf element, the corresponding transition X is renamed to ‘produce X’ (STEP 6).
Else, if data element X is not a leaf element, the corresponding transition X is replaced by a ‘prepare X’ and a ‘produce X’ transition (STEP 2). Next, if data element X is produced by more than one operation, a transition for each operation is created and one input place and one output place to these transitions are added and connected to each other. The ‘prepare X’ transition is connected to the input place, and the output place is connected to the ‘produce X’ transition (STEP 3). Then, if the operation OpX only has one data input element, the corresponding transition is renamed to the input element. Else, if the operation has several data input elements, a ‘prepare OpX’ and a ‘produce OpX’ transition are added and connected (STEP 4) and, an input place, a transition, and an output place are created for each input data element and connected to the ‘prepare OpX’ and ‘produce OpX’ transitions (STEP 5).
These steps are repeated for all data elements that are input to operations producing X, as if the input data element was a new root element (STEP 6).
Finally, a simplification rule is applied to reduce the size of the resulting Petri net. Each part of the model which is of the form of a single input place connected to a ‘prepare’ transition connected to a single output place, can be replaced by just one place (STEP 7).
Problems:
1. When an input data element is used in several operations, a transition to produce this data element will occur several times in the process model. In case of a choice construct this does not lead to any problems, since only one of the alternative paths will eventually be followed and the transition will only be executed once (as in the example). However, in case the data element is needed more than once for determination of the root element, this results in double determination of the same data element.

Solutions:
1. Problem 1 can be solved by assuming that a particular data element can only be produced once - and that therefore only one of the transitions determining it should be able to fire. Once a data element is produced, the other transitions producing the same data element should be skipped when enabled. In order to obtain this situation some extra administrative counter-places and skip-transitions should be added to the process model. But then we also need to cancel the tokens which are left behind when the end product is determined in order to obtain a sound model. This can only be done with a higher modeling language such as YAWL (see algorithm Delta).

Correct translation of the PDM:
In case a data element is present in more than one alternative path to the root element, or in case a data element is needed more than once for determination of the root element, the process model contains several occurrences of the same transition. In the latter situation this means that the value for a data element can be produced more than once, which may not be a correct translation of the PDM.

Soundness of the process model [8]:
The algorithm provides a sound process model. For every token in the start place there will arrive exactly one token in the end place and no tokens are left in the model.
3.2 from PDM to PM, algorithm Bravo

Classification:

<table>
<thead>
<tr>
<th>Representation</th>
<th>Petri nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order</td>
<td>Top-down</td>
</tr>
<tr>
<td>Focus</td>
<td>Operations</td>
</tr>
<tr>
<td>Moment of choice</td>
<td>Early</td>
</tr>
<tr>
<td>Eagerness</td>
<td>Strict</td>
</tr>
<tr>
<td>Concurrent execution of steps</td>
<td>Possible</td>
</tr>
</tbody>
</table>

Description of the algorithm:

Summary: All alternative paths are elaborated top-down from the root element. The details of how to produce a data element are added to the process model, working from both the input and output place. The different paths are elaborated from root to leaf elements one-by-one.

Step-by-step: Start with just one input (i) and one output (o) place (STEP 1). The output place represents a data element: the root element. Now, select one of the operations producing this root element. A transition is created for this operation and connected to the output place.

Based on the number of input elements to the operation, one of three steps is executed:

1. if the number of input data elements to this operation is zero, the input place 'i' is connected to the transition. (STEP 3, STEP 6, STEP 8)
2. if the number of input data elements is one, an input place 'pi' to the transition is added and connected to the transition. Then these steps (STEP 1-3) are repeated for input place 'i' and output place 'pi'. (STEP 2)
3. if the number of input data elements is more than one, an initializing transition is added and the input place 'i' is connected to this transition (STEP 4). Two places 'p1' and 'p2' are added for every input data element and the initializing transition is connected to 'p1', and 'p2' is connected to the transition for the operation (STEP 5, STEP 7). Then, these steps (STEP 1-3) are repeated with input place 'p1' and output place 'p2'.

These steps are repeated for all operations producing the root element (STEP 2-3 – STEP 4-8).

Note that the selection of operations is arbitrary. Thus, STEP 2-3 could be interchanged with STEP 4-8. For similar reasons also STEP 5-6 could be interchanged with STEP 7-8.

Problems:

1. When an input data element is used in several operations (as C in the example), a transition to produce this data element will occur several times in the process model (as transition O4 in the example). In case of a choice construct this does not lead to problems, since only one of the alternative paths will eventually be followed and the transition will only be executed once (as in the example). However, in case the data element is needed more than once for determination of the root element, this results in a double determination of the same data element (see “test14 ABCDEEF” in Appendix A).
Solutions:
1. Problem 1 can be solved by assuming that a particular data element can only be produced once - and that therefore only one of the transitions determining it should be able to fire. Once a data element is produced, the other transitions producing the same data element should be skipped when enabled. In order to obtain this situation some extra administrative counter-places and skip-transitions should be added to the process model. But then we also need to cancel the tokens which are left behind when the end product is determined in order to obtain a sound model. This can only be done with a higher modeling language such as YAWL (see algorithm Delta).

Correct translation of the PDM:
In case a data element is present in more than one alternative path to the root element, or in case a data element is needed more than once for determination of the root element, the process model contains several occurrences of the same transition. In the latter situation this means that the value for a data element can be produced more than once, which may not be a correct translation of the PDM.

Soundness of the process model [8]:
The algorithm provides a sound process model. For every token in the start place, eventually there will be exactly one token in the end place.
3.3 from PDM to PM, algorithm Charlie

Classification:

<table>
<thead>
<tr>
<th>Representation</th>
<th>Petri nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order</td>
<td>Middle-out</td>
</tr>
<tr>
<td>Focus</td>
<td>Data elements</td>
</tr>
<tr>
<td>Moment of choice</td>
<td>Late</td>
</tr>
<tr>
<td>Eagerness</td>
<td>Overcomplete and unrestricted</td>
</tr>
<tr>
<td>Concurrent execution of steps</td>
<td>Possible</td>
</tr>
</tbody>
</table>

Description of algorithm:

Summary: First, all data elements of the PDM are translated to a component consisting of an input place, a transition and an output place. Next, these components are connected in the right way and an initializing transition is added.

Step-by-step: For every data element a transition and an input and output place to this transition are created (STEP 1). Next, these parts are connected with each other through silent transitions for each operation. The input places to a silent transition are the output places of the transitions producing the input elements of the operation. The output place of the silent transition corresponds to the input place of the transition producing the output element of the operation (STEP 2). Finally, a start place and initializing transition are added and connected to the input place of all transitions producing leaf elements (STEP 3). The initializing transition is an AND-split, such that all transitions producing a leaf element are enabled.

Problems:

1. When an input data element is used in several operations, it is only represented once in the process model. This means that the transition to produce the data element can only be executed once (which is good!), but the result then can also only be used once (which is undesirable!). See for instance "test14 ABCDEEF" in Appendix A; the end product will never be produced.

2. When several alternative branches lead to a data element, all of the branches are enabled initially. This means that the data element can be determined multiple times. This data element could be any non-leaf element, including the root element. See "test06 ABCDor" in Appendix A.

Solutions:

1. Put a token back to an "out_X" place when the token was used by a silent transition. However, this solution will result in (more) tokens remaining in the Petri net after completion.

2. Add an administration layer (extra places and transitions so the model obtains certain properties) that uses "control-places" that make sure a transition only fires once to get a sound model. To obtain a lazy sound model [9] it is enough to add an administration layer such as the one used in algorithm Echo.

To incorporate these solutions in a way that still leaves a reasonable model, a higher modeling language such as YAWL should be used, which includes cancellation regions/cancellation activities (see algorithm Delta).

Correct translation of the PDM:

Since the result of the production of a data element can only be used once (see problem 1), this is not a completely correct representation of the PDM.
Soundness of the process model [8]:
The process model is not sound because at the start of the process all possible branches leading to the end product are enabled. This means either the end product (and possibly also intermediate products) can be produced multiple times (see "test06 ABCDor" in Appendix A), or that some tokens are left in the model after determining the root element (see "test08 ABCCD" in Appendix A).


### 3.4 from PDM to PM, algorithm Delta

**Classification:**

<table>
<thead>
<tr>
<th>Representation</th>
<th>YAWL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order</td>
<td>Middle-out</td>
</tr>
<tr>
<td>Focus</td>
<td>Data elements</td>
</tr>
<tr>
<td>Moment of choice</td>
<td>Late</td>
</tr>
<tr>
<td>Eagerness</td>
<td>Overcomplete, but restricted</td>
</tr>
<tr>
<td>Concurrent execution of steps</td>
<td>Possible</td>
</tr>
</tbody>
</table>

**Description of the algorithm:**

**Summary:** First, all data elements of the PDM are translated to a component consisting of an input place, a transition and an output place. Next, these components are connected in the right way and an initializing transition is added. To make it a sound process model, control places (assuring that each transition can only be enabled once) and cancellation arcs (cleaning the net when the end product is produced) are added.

**Step-by-step:**

1. For every data element a transition and an input and output place to this transition are created (STEP 1).
2. Next, these parts are connected with each other through silent transitions for each operation. The input places to a silent transition are the output places of the transitions producing the input elements of the operation. The output place of the silent transition corresponds to the input place of the transition producing the output element of the operation. Moreover, the silent transition is connected to each of its input places (in two directions) (STEP 2).
3. Then, a start place and initializing transition are added and connected to the input place of all transitions producing leaf elements (STEP 3). The initializing transition is an AND-split, such that all transitions producing a leaf element are enabled.
4. Next, control places are created for every transition and connected to the corresponding transition to make sure the transition can only fire once (STEP 4).
5. Finally, a cancellation region is added to the transition producing the root element (STEP 5).

**Problems:**

We have not encountered any problems with this algorithm.

**Correct translation of the PDM:**

The process model gives a correct representation of the PDM. A data element can only be produced once, but it can be used in several operations.

**Verification of the process model [6]:**

For YAWL models there are four desirable properties. These are *soundness*, *weak soundness*, *irreducible cancellation regions* and *immutable OR-joins*. Since the generated process model does not contain any OR-joins, the model has last mentioned property.

**Soundness [6]:**

The *soundness* property actually is a combination of three desirable properties: *option to complete*, *proper completion* and *no dead transitions*. The generated process model fulfills the *option to complete* and the *no dead transitions* property, but not the *proper completion* property. However, due to the use of cancellation regions, the process model is structured in such a manner that the same effect as *proper completion* is obtained.
Weak soundness [6]:
The weak soundness property is, compared to soundness, a relaxation of the option to complete property. For this relaxation is no need since the process model has the option to complete property.

Irreducible cancellation regions [6]:
The process models that are generated from a PDM also fulfill the irreducible cancellation regions property.

Remarks:
Because of the extra cancellation arcs and control places the readability of the model is generally bad.

For the usage of YAWL specific tools are needed. See [11] for more information.
3.5 from PDM to PM, algorithm Echo

Classification:

<table>
<thead>
<tr>
<th>Representation</th>
<th>Petri nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order</td>
<td>Middle-out</td>
</tr>
<tr>
<td>Focus</td>
<td>Operations</td>
</tr>
<tr>
<td>Moment of choice</td>
<td>Late</td>
</tr>
<tr>
<td>Eagerness</td>
<td>Overcomplete and unrestricted</td>
</tr>
<tr>
<td>Concurrent execution of steps</td>
<td>Possible</td>
</tr>
</tbody>
</table>

Description of the algorithm:

Summary: First, for each data element a component that consists of an input place connected to a transition is created. Then, for each operation a transition is added with the right number of input places (i.e. one for every input data element). The transitions corresponding to a data element are connected to these input places in the right way. Finally, an extra control place is added to assure that the end product can not be produced more than once.

Step-by-step: An input place, an output place and an initializing transition are created. The input place is connected to the initializing transition (STEP 1).
Next, for every data element a place and a transition are created. The place is connected to the transition (STEP 2).
Then, one operation is selected. A transition ‘O’ is added for this operation. The transition O is connected to the input place of the transition corresponding to the output element of the operation. Also, a number of input places to transition ‘O’ are added. For each input element of the operation one input place to the transition is created and connected. In case the operation has no input elements one input place is created. Finally, the transitions corresponding to the input elements of the operation are connected to the corresponding input place (STEP 3). This step is repeated for all operations in the PDM (STEP 4, STEP 5, STEP 6).
Finally, an extra place is added to ensure that the root element can be produced only once. The initializing transition is connected to this place and this place is connected to the transition producing the root element (STEP 7).
Note that STEP 3-6 can be executed in any arbitrary order.

Problems:

1. When several alternative branches lead to a data element, all of the branches are enabled initially. This means that the data element might be determined multiple times.

Solutions:

1. For the root element an extra administration layer can be added to the model. This layer can for instance act like a vacuum cleaner: as soon as the root element is produced, all other tokens are removed from the model. A higher modeling language, such as YAWL, should be used, since it includes cancellation regions/cancellation activities. See algorithm Delta for such a vacuum cleaner. Another possible administration layer could enforce a different kind of soundness, for instance lazy soundness (such as the layer used). Lazy soundness means that the end place is reached exactly once. After the end place is reached, other tokens might be remaining in the model and some transitions might still fire [9]. Please note that this solution does not solve the problem for elements other than the root element!
*Correct translation of the PDM:*

The resulting process model is a correct representation of the PDM.

*Soundness of the process model [8]:*

However, the process model is not sound because at the start of the process all possible branches leading to the end product are enabled. This means that, if the number of branches is more than one, some tokens are left in the model after executing the last step.

With the selected administration layer, the resulting model is lazy sound. Please note that with another administration layer, some other soundness (e.g. ‘normal’ soundness or relaxed soundness) might be established, but these will become very complex structures.
3.6 from PDM to PM, algorithm Foxtrot

Classification:

<table>
<thead>
<tr>
<th>Representation</th>
<th>Petri nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order</td>
<td>Bottom-up</td>
</tr>
<tr>
<td>Focus</td>
<td>Operations</td>
</tr>
<tr>
<td>Moment of choice</td>
<td>Late</td>
</tr>
<tr>
<td>Eagerness</td>
<td>Overcomplete and unrestricted</td>
</tr>
<tr>
<td>Concurrent execution of steps</td>
<td>Not possible</td>
</tr>
</tbody>
</table>

Description of the algorithm:

Summary: This algorithm starts with the leaf elements of the PDM. A place represents the available data elements. Starting from the input place of the model with no data elements available the algorithm computes all possible steps that can be executed.

Step-by-step: Start with just one input place and one output place (STEP 1). Then, select a leaf element. A transition and an output place for this transition are created and connected. Also, the input place is connected to the transition (STEP 2).

Now a place represents the available data elements. From the output place of the transition we start reasoning further. One of the operations of which all input elements are available and which is not executed yet is selected. For this operation a transition is added and connected to a newly created place if this place was not existing yet (STEP 3, STEP 4, STEP 6, STEP 8). Otherwise the transition is connected to the already existing place (STEP 5, STEP 7, STEP 9).

If no further steps are possible from the place we have reached we look back whether another branch can be started. This branch is also elaborated step-by-step as described above (STEP 5, STEP 8).

These steps are executed for all leaf elements (STEP 6).

Finally, all places containing the root element are connected with the end place via a silent transition (STEP 10).

Problems:

1. Because of the exponential blowup (see Remarks below) computation time and space may become a problem. This depends on the size of the model and the system on which the algorithm is run.

Correct translation of the PDM:
The process model is a correct representation of the PDM.

Soundness of the process model [8]:
The process model is a sound model.
Remarks:
Note that a bottom-up approach to construct a (static) process model from a PDM will create an exponential blowup in the number of places and transitions, as clearly can be seen in this algorithm. Readability and understandability rapidly decrease as the number of places and transitions grow! The bottom-up construction strategy is more suitable for a more dynamic engine approach, in which the executable steps regularly are calculated based on the available information.

The process models produced by this algorithm lean very close towards visualizing the process run-time, so one could even argue that this is not a model, but a visualization of all possible runs of the ‘super’-model; i.e. it describes the statespace of all possible execution paths.

The execution of the PDM is not restricted in any way. Each time all possible steps can be chosen for execution, with the remark that, although an operation may be represented more than once in the process model as a transition, only one of these transitions may be executed on a single run of the process model!
In this process model there will only be one token at all time, so parallel execution of transitions is not possible.
3.7 from PDM to PM, algorithm Golf

Classification:

<table>
<thead>
<tr>
<th>Representation</th>
<th>Petri nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order</td>
<td>Mix of: top-down and bottom-up</td>
</tr>
<tr>
<td>Focus</td>
<td>Operations</td>
</tr>
<tr>
<td>Moment of choice</td>
<td>Late</td>
</tr>
<tr>
<td>Eagerness</td>
<td>Strict</td>
</tr>
<tr>
<td>Concurrent execution of steps</td>
<td>Not possible</td>
</tr>
</tbody>
</table>

Description of the algorithm:

Summary: All possible execution paths to produce the end product are determined top-down. Next, each path is elaborated bottom-up, starting with the first element that should be produced from the start. Each place is representing the available data elements. A path is build from the place containing the available data elements. A transition representing an operation that can be executed from this place is added. This transition produces a new data element which is reflected in a new place following the transition. Again the next transition corresponds to an operation that can be executed based on the available data elements, etcetera. Each time before a place is created it is checked first whether the right place already exists to avoid duplicate places. In case the right place already exists the transition is connected to this place instead of introducing a new place.

Step-by-step: First all possible sets of operation executions, strictly leading to the end product, are calculated.

A start place and an end place are added to the process model (STEP 1). Then, one of the paths is selected and build bottom up. A first step in the path is selected and a transition and output place of this transition are created. Also, the input place of the model is connected to the transition (STEP 2). Then, a next step in the execution sequence is selected. Again, a transition is added for this operation. Also, an output place, representing the available data elements is added and the transition is connected to this place. If no further steps are possible from the place we have reached (STEP 3) we look back the path whether another branch can be started on the current path. These steps are repeated until the path is finished (STEP 4). While repeating these steps the existence of transitions and places is checked. If a place representing a number of available data elements or an overlapping part of the path was already created before, this component in the model is used instead of creating a new place.

When the path is finished a silent transition connecting the final place of the path with the end place of the process model is added (STEP 5). Then, these steps are repeated for all paths (STEP 2-5, STEP 6-7).

Problems:
1. Because of the exponential blowup (see Remarks below) computation time and space may become a problem. This depends on the size of the model and the system on which the algorithm is run.

Correct translation of PDM:
The process model is a correct representation of the PDM.

Soundness of the process model [8]:
The resulting process model is sound.
Remarks:
The translation has an exponential blowup in places and transitions, although in most cases not as bad as the Foxtrot algorithm, since the statespace is restricted to the paths in the PDM and hence not always completely calculated. Readability and understandability rapidly decrease as the number of places and transitions grow!

Please note that, although an operation may be represented more than once in the process model as a transition, only one of these transitions may be executed on a single run of the process model! In this process model there will only be one token at all time, so parallel execution of transitions is not possible.
Execution paths: \{\{O1, O3, O4\}, \{O2, O4\}\}
4 Related work
A first step to automate the generation of process models is made in [3].
In order to express the product structure of products without physical manifestation in a
better way than can be done with a Bill-of-Material, the PDM was introduced in [2].
Application of the PDM, mostly theoretical, can be found in [2], [4] and [5].

As described in [1], the area of business process redesign and more specifically product
based workflow design covers a much wider area than the scope of this work. For more
related literature we refer to the second chapter of [1].

The algorithms as described above are implemented in the open source ProM Framework
for process mining. For more information, see [12].

5 Conclusions and future work
In this paper we described and classified seven algorithms that can be used for deriving
process models based on a product data model. Some of the algorithms still possess
problems, which might need to be resolved.
Given a specific situation, the classification can be used to determine which algorithm will
deliver the best process model for that situation. It is up to the end user and/or process
modeler to make this consideration. If an end user or a process modeler has criteria or
wishes we did not include in the classification, a need for other algorithms might arise.
It would be nice to extend the PDM with specific data attributes (e.g. cost of an
operation, time before an operation completes, failure chance of an operation). The
extension of the PDM with these data attributes will improve its usability in realistic
situations. Also, this extension will probably also give rise to the need for a greater
variety of algorithms.
There is a need for more practical uses of the PDM. Right now there are only a few cases,
e.g. some simple examples in our benchmark set (see appendix A), and the GAK-case
which can be found in [2].
References

[1] "Intelligent" tools for workflow process redesign: A research agenda
M. Netjes, I. Vanderfeesten, H.A. Reijers
Lecture Notes in Computer Science, Volume 3812, 2006 – pages 444 to 453

H.A. Reijers
Dissertation, 2002
Also available as book 2617 in the Lecture Notes in Computer Science series.
Publisher: Springer; 1st edition (July 29, 2003)

W.M.P. van der Aalst
Computers in Industry 39, 1999 – pages 97 to 111

[4] Product Based Workflow Design with Case Handling Systems
I. Vanderfeesten, H.A. Reijers, W.M.P. van der Aalst

[5] Modelling a product based workflow system in CPN tools
I. Vanderfeesten, W.M.P. van der Aalst, H.A. Reijers

M.T. Wynn, H.M.W. Verbeek, W.M.P. van der Aalst, A.H.M. ter Hofstede, D. Edmond

[7] YAWL: Yet Another Workflow Language
W.M.P. van der Aalst, A.H.M. ter Hofstede
Information Systems, 30(4), 2005 – pages 245 to 275

K. Barkaoui, R. Ben Ayed, Zohra Sbaï
International Journal of Computing & Information Sciences, Volume 5, No. 1, April 2007 – pages 51 to 61

[9] Investigations on Soundness Regarding Lazy Activities
F. Puhlmann, M. Weske
Lecture Notes in Computer Science, Volume 4102, 2006 – pages 145 to 160

[10] Product-Based Workflow Design
H.A. Reijers, S. Liman, W.M.P. van der Aalst

http://www.yawl-system.com/

[12] ProM website
http://prom.sourceforge.net/
## Appendix A: Benchmark Set

<table>
<thead>
<tr>
<th>Benchmark Test</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>test01 AB</td>
<td>from PDM to PM, algorithm Bravo</td>
</tr>
</tbody>
</table>

### from PDM to PM, algorithm Alpha

```
produce_B  →  produce_A
```

### from PDM to PM, algorithm Echo

```
T_(B, 1)  →  T_(A, (B))
```

### from PDM to PM, algorithm Charlie

```
T_(A, B)  →  T_(A, B)
```

### from PDM to PM, algorithm Golf

```
T_(B, 0)  →  T_(A, (B))  →  T_
```

### from PDM to PM, algorithm Foxtrot

```
T_(B, 0)  →  T_(A, (B))  →  T_
```
From Product Data Model to Process Model

Johfra Kamphuis, Irene Vanderfeesten, Hajo A. Reijers, Maarte van Hattem
From Product Data Model to Process Model

From PDM to PM, algorithm Bravo

From PDM to PM, algorithm Alpha

From PDM to PM, algorithm Echo

From PDM to PM, algorithm Charlie

From PDM to PM, algorithm Golf

From PDM to PM, algorithm Delta

From PDM to PM, algorithm Foxtrot
From Product Data Model to Process Model
Johfra Kamphuis, Irene Vanderfeesten, Hajo A. Reijers, Maarte van Hattem
test05 ABCC

from PDM to PM, algorithm Bravo

from PDM to PM, algorithm Alpha

from PDM to PM, algorithm Echo

from PDM to PM, algorithm Charlie

from PDM to PM, algorithm Golf

from PDM to PM, algorithm Delta

from PDM to PM, algorithm Foxtrot
From Product Data Model to Process Model

Johfra Kamphuis, Irene Vanderfeesten, Hajo A. Reijers, Maarte van Hattem

**test06 ABCDor**

- From PDM to PM, algorithm Bravo

**from PDM to PM, algorithm Alpha**

**from PDM to PM, algorithm Echo**

**from PDM to PM, algorithm Charlie**

**from PDM to PM, algorithm Golf**

**from PDM to PM, algorithm Delta**

**from PDM to PM, algorithm Foxtrot**
From Product Data Model to Process Model
Johfra Kamphuis, Irene Vanderfeesten, Hajo A. Reijers, Maarte van Hattem

From PDM to PM, algorithm Bravo

From PDM to PM, algorithm Alpha

From PDM to PM, algorithm Echo

From PDM to PM, algorithm Charlie

From PDM to PM, algorithm Golf

From PDM to PM, algorithm Delta

From PDM to PM, algorithm Foxtrot
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bravo</td>
<td>From PDM to PM, algorithm Bravo</td>
</tr>
<tr>
<td>Alpha</td>
<td>From PDM to PM, algorithm Alpha</td>
</tr>
<tr>
<td>Echo</td>
<td>From PDM to PM, algorithm Echo</td>
</tr>
<tr>
<td>Charlie</td>
<td>From PDM to PM, algorithm Charlie</td>
</tr>
<tr>
<td>Golf</td>
<td>From PDM to PM, algorithm Golf</td>
</tr>
<tr>
<td>Foxtrot</td>
<td>From PDM to PM, algorithm Foxtrot</td>
</tr>
</tbody>
</table>
From Product Data Model to Process Model

Johfra Kamphuis, Irene Vanderfeesten, Hajo A. Reijers, Maarte van Hattem
From Product Data Model to Process Model
Johfra Kamphuis, Irene Vanderfeesten, Hajo A. Reijers, Maarte van Hattem

from PDM to PM, algorithm Bravo

from PDM to PM, algorithm Alpha

from PDM to PM, algorithm Echo

from PDM to PM, algorithm Charlie

from PDM to PM, algorithm Golf

from PDM to PM, algorithm Delta

from PDM to PM, algorithm Foxtrot
From Product Data Model to Process Model
Johfra Kamphuis, Irene Vanderfeesten, Hajo A. Reijers, Maarte van Hattem
From Product Data Model to Process Model

Johfra Kamphuis, Irene Vanderfeesten, Hajo A. Reijers, Maarte van Hattem
From Product Data Model to Process Model
Johfra Kamphuis, Irene Vanderfeesten, Hajo A. Reijers, Maarte van Hattem