Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
— Institut im Quellcode anpassen nicht vergessen! —

Transforming Object-Aware Processes into BPMN: Conceptual Design and Implementation

Abschlussarbeit an der Universität Ulm

Vorgelegt von:
Marko Pejic
marko.pejic@uni-ulm.de
1027682

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Marius Breitmayer

2023

Fassung 14. November 2022

Satz: PDF-LaTeX 2_ε

Inhaltsverzeichnis

1 Introduction

1.1 Objective

1.2 Problem Statement

1.3 Structure of Thesis

2 Fundamentals

2.1 Business Process Management

2.2 Process Modeling

2.2.1 Activity-Centric

2.2.2 Data-Centric

2.3 Business Process Model and Notation

2.4 Object-Aware Process Management

3 Requirements

3.1 Functional Requirements

3.2 Non-Functional Requirements

4 Transforming Object-Aware Processes into BPMN

In this chapter, a conceptual transformation model is proposed that fulfills the defined requirements for the transformation of object-aware processes into BPMN (cf. Chap. 3). First, an overview of the necessary transformations and mapping rules for the mentioned conception is given (cf. Sect. 4.1). Next, the concrete mapping (cf. Sect. 4.2) and another component of the conception - Robotic Process Automation (RPA) (cf. Sect. 4.3) - are explained in more detail. Finally, an algorithm for transforming object-aware processes into BPMN can be developed as a result (cf. Sect. 4.4).

4.1 Transformation Compendium

The transformation model is divided into three different types: Object, Permission, and Relation, with each type providing different functionalities to fulfill a different concern in the transformation model. In addition, an external component completes the latter.

Type 1 (Object Transformation)**.** *Informell sagen was passiert.*

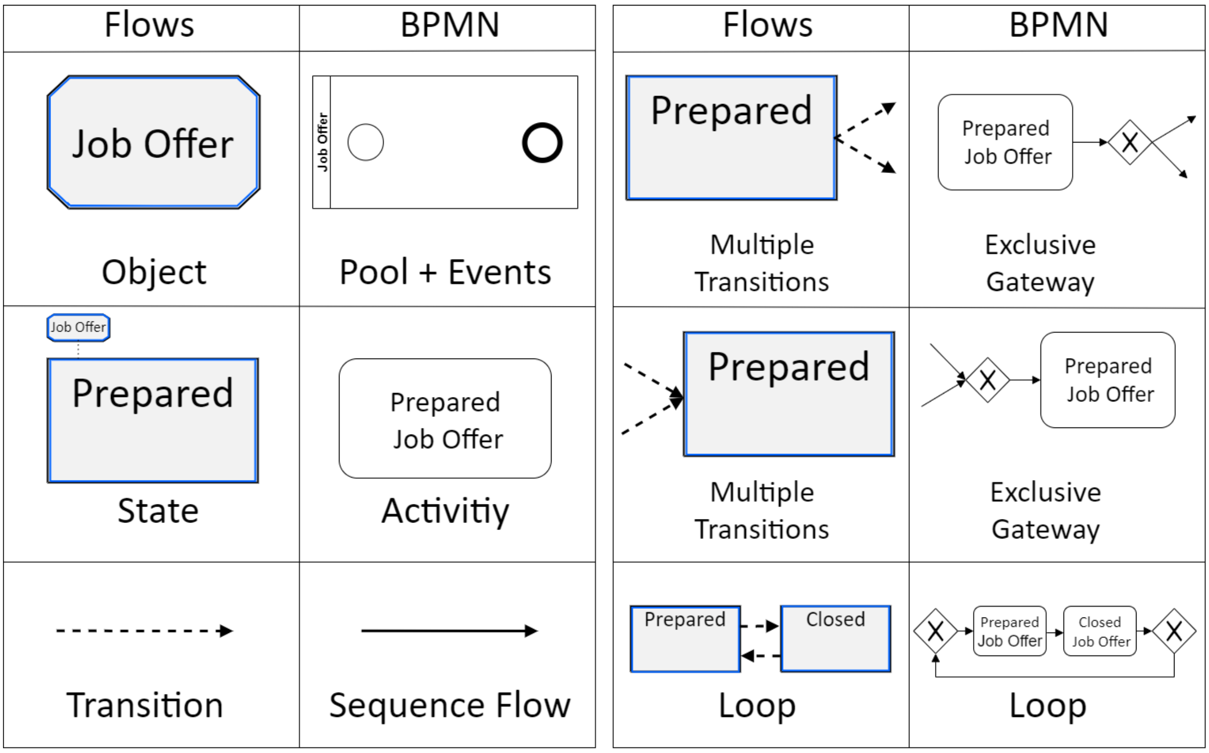

Abbildung 4.1: Initial example

Type 2 (Permission Integration)**.** *Informell sagen was passiert.*

Type 3 (Relation Integration)**.** *Informell sagen was passiert.*

4.2 Object Transformation

The object transformation deals with the task of transforming an object in terms of BPMN. For this purpose, the individual elements, i.e., states, steps, and transitions, that occur in the lifecycle of the object, are transformed individually. Furthermore, other BPMN-related elements are included in the object transformation, namely pools, events, gateways, and data objects.

Definition 1 (Object Transformation)**.** *Let* $\omega = (\mathrm{n}_\omega, \Phi, \Theta)$ *be an object. Then: function* $OTA : \Omega \rightarrow \mathrm{P}$ *generates a pool* $\rho = (\mathrm{n}_\rho, \delta)$*, where* $\delta = (\mathrm{E}, \mathrm{A}, \mathrm{T}_\delta, \mathrm{K}, \mathrm{N}, \Xi)$ *is a 6-tuple of BPMN process elements within the pool.*

Informally stated, OTA receives an object as input, along with its name, lifecycle, and a set of attributes, and returns a pool with its name and a 6-tuple containing its BPMN process elements corresponding to the object, i.e., a set of events, a set of activities, a set of transitions, a set of gateways, and a set of data objects. At this step of transformation, lanes are not considered yet, hence, by default the set of lanes is empty during object transformation. Algorithm 1 describes the generation of these components in pseudo code, partly using separate functions, where each function transforms a different component. Later in this section, these functions are explained in more detail.

Algorithm 1 Object Transformation Algorithm (OTA)

Require: $\omega = (\mathrm{n}_\omega, \Phi, \Theta)$
  $\rho, \mathrm{E}_\delta, \mathrm{A}, \mathrm{T}_\delta, \mathrm{N}, \Xi \leftarrow \perp$
  $\rho.\mathrm{n}_\rho = \omega.\mathrm{n}_\omega$
  $\mathrm{E}_\delta.add(\epsilon_\delta^{start})$
  $\mathrm{E}_\delta.add(\epsilon_\delta^{end})$
  for all $\sigma\ in\ \Theta.\Sigma$ **do**
    $\alpha \leftarrow AGA(\sigma, \omega)$
    $\mathrm{A}.add(\alpha)$
  end for
  for all $\tau\ in\ \Theta.\mathrm{T}_\omega$ **do**
    $\tau_\delta \leftarrow FTTA(\tau_\omega)$
    $\mathrm{T}_\delta.add(\tau_\delta)$
  end for
  for all $\psi\ in\ \Theta.\Psi$ **do**
    $\tau_\delta \leftarrow BTTA(\psi)$
    $\mathrm{T}_\delta.add(\tau_\delta)$
  end for
  for all $\alpha\ in\ \mathrm{A}$ **do**
    $\tau_\delta \leftarrow ESFA(\alpha)$
    if $\tau_\delta \neq \perp$ **then**
      $\mathrm{T}_\delta.add(\tau_\delta)$
    end if
    $\mathrm{K} \leftarrow GGA(\mathrm{T}_\delta)$
    $\nu \leftarrow DGA(\alpha)$
    $\mathrm{N}.add(\nu)$
  end for
  $\delta \leftarrow (\mathrm{E}, \mathrm{A}, \mathrm{T}_\delta, \mathrm{K}, \mathrm{N}, \Xi)$
  $\rho \leftarrow (\mathrm{n}_\rho, \delta)$
  return ρ

However, some components are not handled by functions, but are predefined at the beginning of the transformation, for the reason, that these components belong to every BPMN process according to the predefined guidelines (cf. TBD). More specifically, at the beginning of the transformation a pool is defined, where the name of the pool corresponds to the name of the respective object. Within the pool, the remaining BPMN elements are defined.

Furthermore, a start and end event within the pool are predefined, since every object has a start and end state. In the course of this section, the transformation is applied step by step onto the object *Job Offer* from Example 1 to illustrate the transformation. Fig. TBD shows the beginning of the transformation, i.e., with generated pool and events, onto the object *Job Offer*.

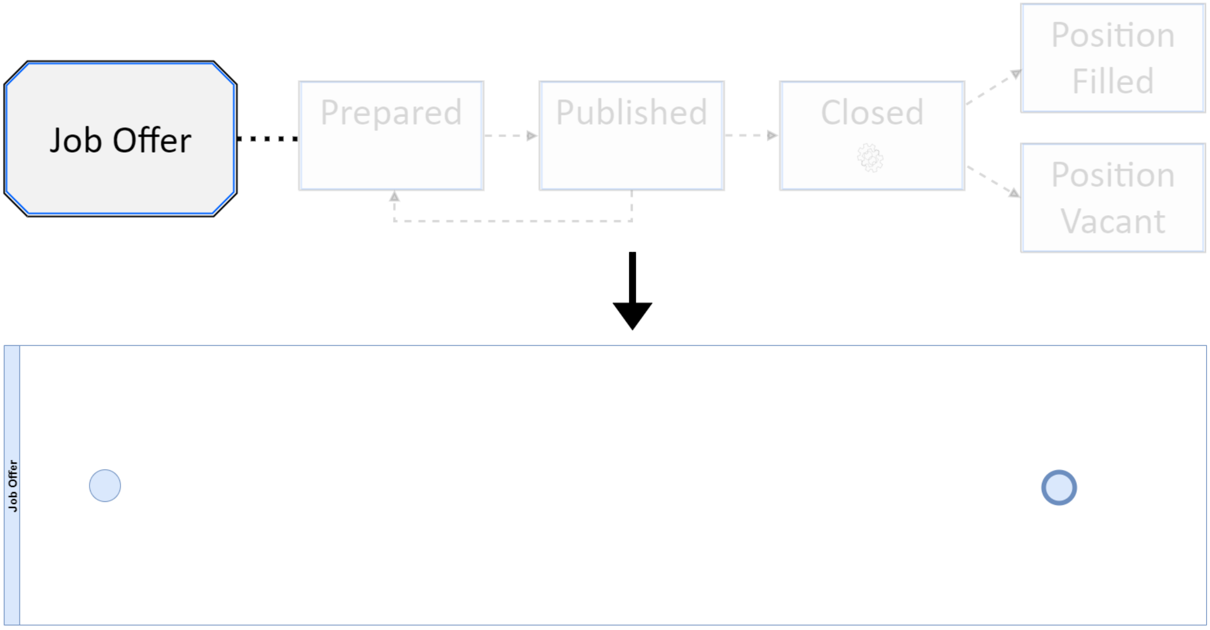

Abbildung 4.2: Initial example

4.2.1 Activity Generation

This transformation step deals with the transformation of states in terms of BPMN. In this context, trivially, each state is transformed into an activity. However, depending on the steps that a state may have, certain cases must be distinguished that lead to one of three types of activities a state can be transformed into:

1. State contains more than one step. In this case, the state is transformed into a *sub-process* (cf. Sect. TBD).
2. State contains exactly one step, and this step is a computational step. In this case, the state is transformed into a *service task*.
3. In any other case, the state is transformed into an activity without any kind of type, i.e., a *non-type activity*.

Furthermore, in any case, the name of an activity results from the name of the state, followed by the name of the object to which the state belongs, separated by a blank character.

Definition 2 (Activity Generation). *Let* ω = $(\mathrm{n}_\omega, \Phi, \Theta)$ *be an object and* Σ *the set of states in the lifecycle* Θ *of* ω*. Then: function* $AGA : \Omega \times \Sigma \rightarrow \mathrm{A}$ *generates the activity* $\alpha = (\mathrm{n}_\alpha, \sigma_\alpha, \iota, SP)$ *from a state* $\sigma = (\mathrm{n}_\sigma, \Gamma, \mathrm{T}_\sigma, \Psi) \in \Sigma$.

Definition 2.1 (Computational Step). *Let* ω = $(\mathrm{n}_\omega, \Phi, \Theta)$ *be an object and* $\gamma \in \Gamma$ *a step from the set of steps* Γ *from a state* $\sigma = (\mathrm{n}_\sigma, \Gamma, \mathrm{T}_\sigma, \Psi) \in \Sigma$ *in the lifecycle* Θ *of* ω*. Then: function* $CS : \Gamma \rightarrow \mathbb{B}$ *determines whether the given step is a computational step.*

$$CS(\gamma) = \begin{cases} true, & \textit{if } \gamma\ .\lambda \neq \perp. \\ false, & \textit{otherwise.} \end{cases}$$

Definition 2.2 (Activity Type Generation). *Let* ω = $(\mathrm{n}_\omega, \Phi, \Theta)$ *be an object and* Γ *the set of steps from a state* $\sigma = (\mathrm{n}_\sigma, \Gamma, \mathrm{T}_\sigma, \Psi) \in \Sigma$ *in the lifecycle* Θ *of* ω*. Then: function* $ATG : \Sigma \rightarrow \mathrm{I}$ *determines the type* ι *of the activity* $\alpha = (\sigma_\alpha, \mathrm{n}_\alpha, \iota, SP)$ *that is derived from the state* σ.

$$ATG(\sigma) = \begin{cases} suprocess, & \textit{if } |\Gamma| > 1. \\ service\ task, & \textit{if } (|\Gamma| = 1) \wedge\ (CS(\Gamma.\gamma) = true). \\ non-type, & \textit{otherwise.} \end{cases}$$

Algorithm 3 describes the generation of activities in pseduo code based on Definition 2, 2.1 and 2.2.

Algorithm 2 Activity Generation Algorithm (AGA)

Require: $\sigma = (\mathrm{n}_\sigma, \Gamma, \mathrm{T}_\sigma, \Psi), \omega = (\mathrm{n}_\omega, \Phi, \Theta)$
  $\mathrm{n}_\alpha \leftarrow \sigma.\mathrm{n}_\sigma + \_ + \omega.\mathrm{n}_\omega$
  $\iota \leftarrow ATG(\sigma)$
  if ι = *subprocess* **then**
    $SP \leftarrow TBD(\sigma)$
  else
    $SP \leftarrow \perp$
  end if
  $\alpha \leftarrow (\sigma, \mathrm{n}_\alpha, \iota, SP)$
  return α

Fig. TBD graphically illustrates how this transformation step can be applied to further transform the object *Job Offer*.

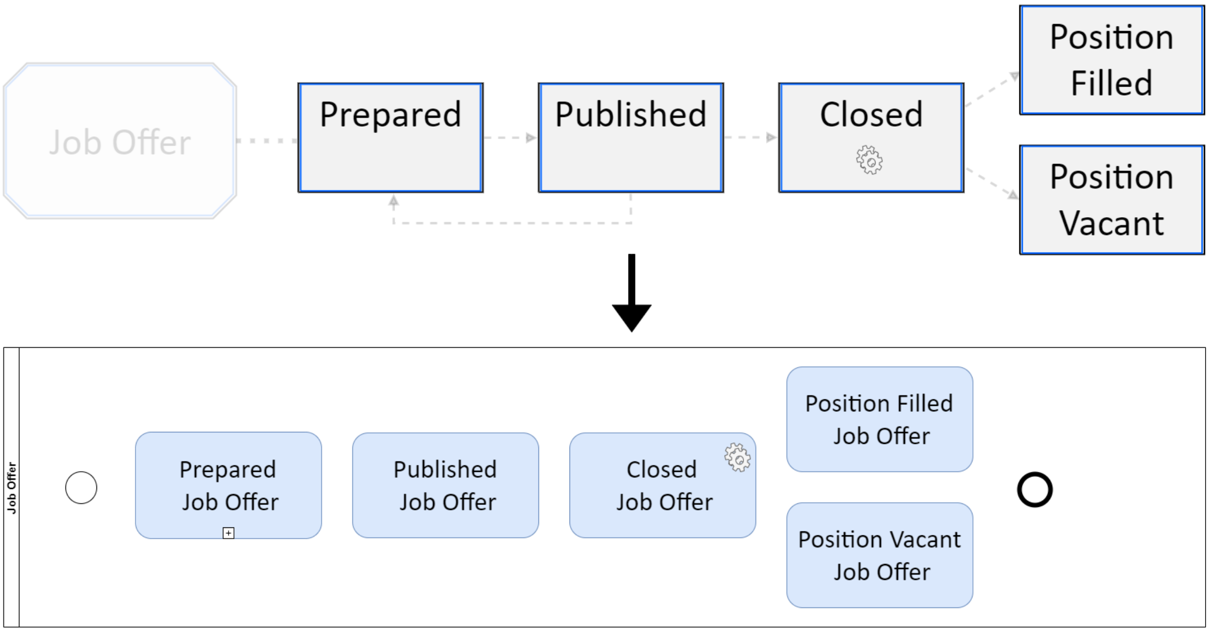

Abbildung 4.3: AGA example

4.2.2 Transition Generation

This transformation step deals with the transformation of transitions from an object in terms of BPMN. In BPMN, transitions between elements can be regulated by sequence flows. Therefore, in this section, an algorithm transforming an object transition into a sequence flow is provided. Two types of object transitions are possible - forwards and backwards transitions. In the former, a transition is made between two steps from one or more states of an object (e.g., *Prepared Job Offer* to *Published Job Offer*), while in the latter, a transition is made between two states of an object (e.g., *Published Job Offer* to *Prepared Job Offer*). In contrast, more than two types of transitions are possible in BPMN, for example, between two activities, between an activity and an event, or between an activity and a gateway. Consequently, generating sequence flows is not as trivial as generating activities. The latter holds especially true for the transformation of forwards transitions. Backwards transitions, however, can be easily transformed by establishing a sequence flow between the two respective activities that were generated from their respective states with $\mathrm{Algorithm2}$ (cf. Sect. 4.2.1) beforehand. On the contrary, since forwards transitions establish transitions between two steps from one or more objects, two different cases must be distinguished in order to transform them properly:

1. Transitions between two steps belonging to the same state. This case implies that the respective state is a subprocess (cf. Sect. 4.2.1 Definition 2.1). Hence, the transformed transition, i.e., the sequence flow, belongs to the respective subprocess of the activity, rather than the BPMN process of the current object itself. Consequently,

the transformation of such a transition is regulated during the transformation of the respective subprocess.

2. Transitions between two steps that belong to different states. In this case, a sequence flow is drawn between the activities generated from the respective states to which the respective steps belong.

Note that sequence flows between the events and activities of the process are still needed to prevent a deadlock in the process. To this end, this transformation step further iterates through each generated sequence flow so far and determines which activitiy occurs neither as source nor as target of a sequence flow. In the former, sequence flows between the respective activities and the end event of the process are generated, whereas in the latter sequence flows from the start event of the process to the respective activities are generated.

Definition 3 (Sequence Flow Generation)**.** *Let* $\omega = (\mathrm{n}_\omega, \Phi, \Theta)$ *be an object. Further, let* T_ω *be the set of forwards transitions and* Ψ *be the set of backwards transitions in the lifecycle* Θ *of* ω*. Last, let* υ *be a BPMN element that is able to have at least one incoming or outgoing sequence flow, i.e., an activity, an event or a gateway). Then:*

Definition 3.1 (Forwards Transitions Transformation). *Function* $FTTA : \mathrm{T}_\omega \rightarrow \mathrm{T}_\delta$ *generates a sequence flow* $\tau_\delta = (\upsilon_{source}, \upsilon_{target})$ *given a forwards transition* $\tau_\omega = (\gamma_{source}, \gamma_{target})$.

Definition 3.2 (Backwards Transitions Transformation). *Function* $BTTA : \Psi \rightarrow \mathrm{T}_\delta$ *generates a sequence flow* $\tau_\delta = (\upsilon_{source}, \upsilon_{target})$ *given a backwards transition* $\psi = (\sigma_{source}, \sigma_{target})$.

Definition 3.3 (Event Sequence Flows). *Function* $ESFA : \mathrm{A} \rightarrow \mathrm{T}_\delta$ *complements* T_δ *by the missing sequence flows between the events and activities. Given an activity* α*, ESF determines whether a sequence flow between* α *and an start event* ϵ_δ^{start} *or an end event* ϵ_δ^{end} *shall be generated.*

Algorithm 3 describes the generation of sequence flows given the set of forwards transitions from an object based on Definition 3.1. Similarly, Algorithm 4 describes such transformation for the set of backwards transitions based on Definition 3.2. Last, Algorithm 5 describes the generation of the remaining sequence flows between events and activities based on Definition 3.3.

Algorithm 3 Forwards Transition Transformation Algorithm (FTTA)

Require: $\tau_\omega = (\gamma_{source}, \gamma_{target})$

$\tau_\delta \leftarrow \perp$

$\upsilon_{source} \leftarrow \mathrm{getActivity}(\gamma_{source}.\sigma)$

$\upsilon_{target} \leftarrow \mathrm{getActivity}(\gamma_{target}.\sigma)$

$\tau_\delta \leftarrow (\upsilon_{source}, \upsilon_{target})$

return τ_δ

Algorithm 4 Backwards Transition Transformation Algorithm (BTTA)

Require: $\psi = (\sigma_{source}, \sigma_{target})$

$\tau_\delta \leftarrow \perp$

$\upsilon_{source} \leftarrow \text{getActivity}(\sigma_{source})$

$\upsilon_{target} \leftarrow \text{getActivity}(\sigma_{target})$

$\tau_\delta \leftarrow (\upsilon_{source}, \upsilon_{target})$

return τ_δ

Algorithm 5 Event Sequence Flows Algorithm (ESFA)

Require: $\alpha = (\text{n}_\alpha, \iota, SP)$

$\tau_\delta \leftarrow \perp$

if $\alpha.\text{isNoTarget}() = true$ **then**

  $\tau_\delta \leftarrow (\epsilon_\delta^{start}, \alpha)$

else if $\alpha.\text{isNoSource}() = true$ **then**

  $\tau_\delta \leftarrow (\alpha, \epsilon_\delta^{end})$

end if

return τ_δ

Fig. TBD graphically illustrates how the generation of sequence flows from object transitions can be applied onto the object *Job Offer*.

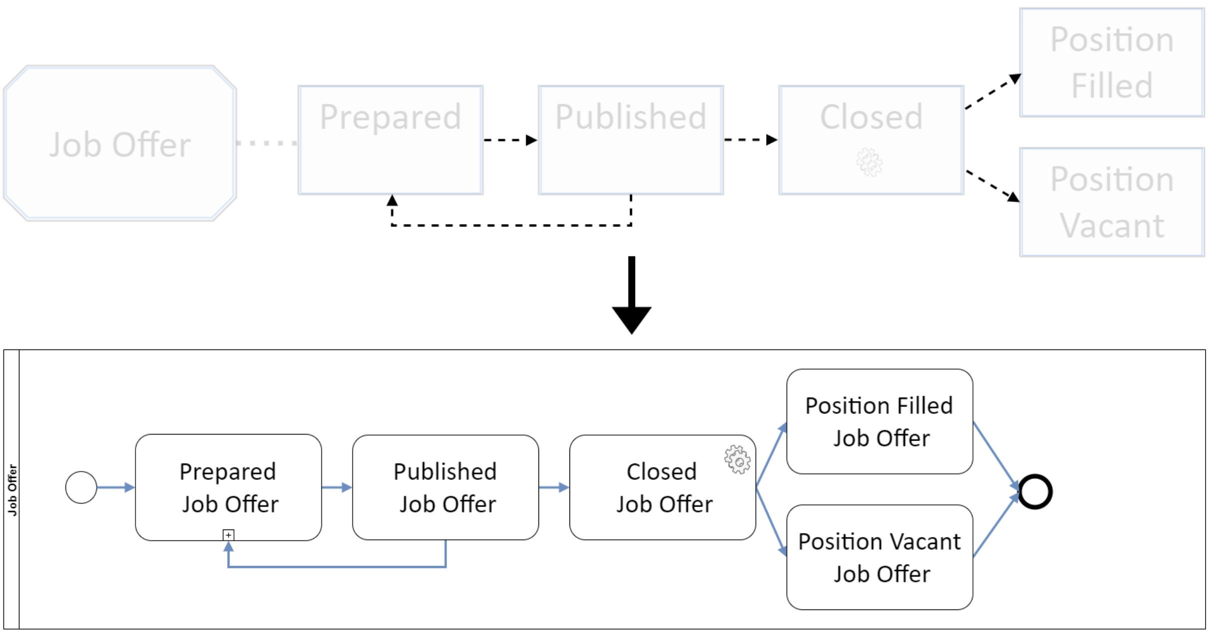

Abbildung 4.4: TGA example

4.2.3 Gateway Generation

At this point in the transformation, activities may have more than one incoming or outgoing sequence flow. However, such circumstances contradict the guidelines for BPMN process modeling (cf. TBD). Therefore, this transformation step is dedicated to the task of resolving these circumstances. For this purpose, exclusive gateways can be used in BPMN, since they may have multiple incoming and outgoing sequence flows. Therefore, for this transformation, each sequence flow is compared to every other one. If multiple, different sequence flows have the same BPMN element as source or target, an exclusive gateway is generated. Furthermore, the sequence flows of the process must be updated so that the respective BPMN element no longer has multiple incoming or outgoing sequence flows, while preserving the logic of the process.

Definition 4 (Gateway Generation)**.** *Let* $\delta = (\mathrm{E}, \mathrm{A}, \mathrm{T}_\delta, \mathrm{K}, \mathrm{N}, \Xi)$ *be a 6-tuple of BPMN process elements derived from an object* $\omega = (\mathrm{n}_\omega, \Phi, \Theta)$*. Further, let* T_δ *be the set of transitions of* δ*. Then: function GGA receives* T_δ *as input to generate exclusive gateways* κ *and return them collected in* K*, i.e., GGA provides* δ *with the set of exclusive gateways* K*. As a side effect, GGA also updates* T_δ *to include or remove the sequence flows related to the exclusive gateways.*

Algorithm 6 describes the generation and integration of exclusive gateways into the current process based on Defintion 4. For the sake of simplicity, the actual procedure of updating T_δ after generating K is not further explained.

Algorithm 6 Gateway Generation Algorithm (GGA)

Require: T_δ
  $\mathrm{K} \leftarrow \perp$
  $N \leftarrow |\mathrm{T}_\delta|$
  for $i \in \{1, ..., N-1\}$ **do**
    $\tau_\delta^i \leftarrow \mathrm{T}_\delta[\mathrm{i}]$
    for $j \in \{i+1, ..., N\}$ **do**
      $\tau_\delta^j \leftarrow \mathrm{T}_\delta[\mathrm{j}]$
      if $\tau_\delta^i.\upsilon_{source} = \tau_\delta^j.\upsilon_{source}$ **then**
        $\kappa \leftarrow \perp$
      end if
      if $\tau_\delta^i.\upsilon_{target} = \tau_\delta^j.\upsilon_{target}$ **then**
        $\kappa \leftarrow \perp$
      end if
      K.*add(*κ*)*
      T_δ.*update()*
    end for

end for
return K

Fig. TBD illustrates the result of this transformation step onto the current process of the object *Job Offer*, i.e., the generation of exclusive gateways where needed and updated transitions.

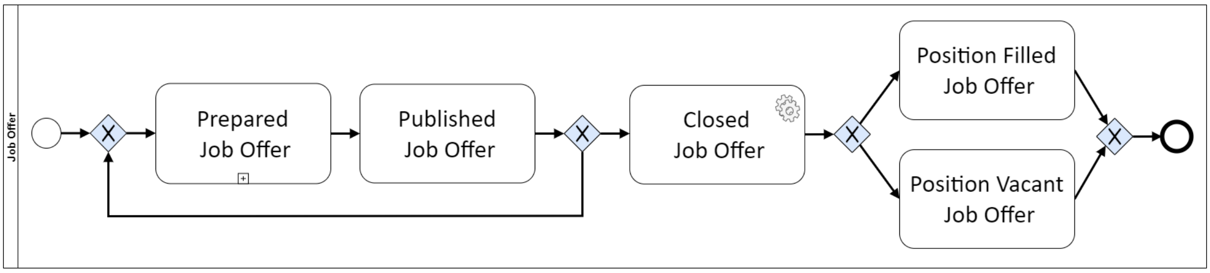

Abbildung 4.5: GGA example

4.2.4 Data Object Generation

Owing to the fact that the execution of the lifecycle process of an object is data-driven, it seems reasonable to consider this integration of data into the BPMN process. In BPMN, data objects can be used for this purpose. This transformation generates a data object for each activity in a process, where each such generated data object shall show, in which state the respective object of the process is in, after the activity in the process has been executed. For this purpose, each such generated data object is labeled with the name of the respective object followed by the name of the state at that point of process execution, with the state enclosed in square brackets. Last, for each such generated data object, it is specified which activities write the data object, i.e., change the state of the object with its execution, and read the data object, i.e., require a certain state of an object or the previous execution of an activity for its execution. However, in this step of the transformation, data objects are only written, while reading data objects is dealt with in a later part of the whole transformation model.

Definition 5 (Data Object Generation)**.** *Let* $\delta = (\mathrm{E}, \mathrm{A}, \mathrm{T}_\delta, \mathrm{K}, \mathrm{N}, \Xi)$ *be a 6-tuple of BPMN process elements derived from an object* $\omega = (\mathrm{n}_\omega, \Phi, \Theta)$*. Further, let* $\alpha \in \mathrm{A}$ *be an activity in the process. Then: function* $DGA : \mathrm{A} \rightarrow \mathrm{N}$ *generates a data object* $\nu = (\mathrm{n}_\nu, \alpha_\nu, \mathrm{P}_{write}, \mathrm{P}_{read})$ *for the respective activity* α*.*

Algorithm 7 shows the generation of data objects in pseudo code based on Definition 5.

Algorithm 7 Data Object Generation Algorithm (DGA)
Require: $\alpha = (\mathrm{n}_\alpha, \sigma_\alpha, \iota, SP)$

$\nu \leftarrow \perp$
$\mathrm{n}_{\nu} \leftarrow \sigma_{\alpha} \ + \ [\mathrm{n}_{\alpha}]$
$\eta \leftarrow \alpha$
$\mathrm{P}_{write}.add(\eta)$
$\nu \leftarrow (\mathrm{n}_{\nu}, \alpha, \mathrm{P}_{write}, \perp)$
return ν

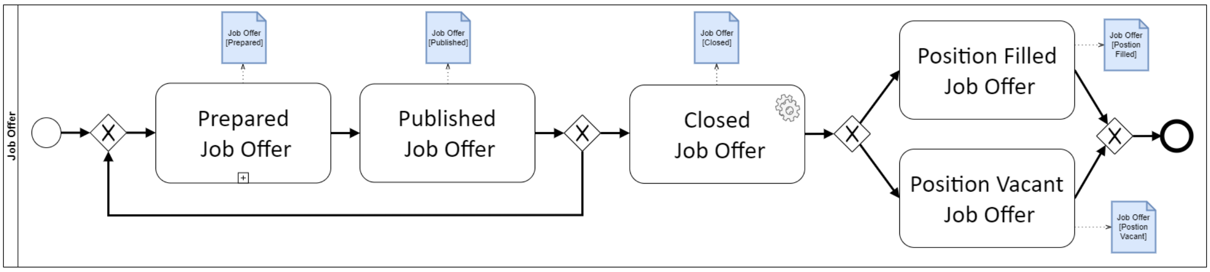

Abbildung 4.6: DOG example

4.2.5 Complementary Transformations

Intro text.

A. Subprocess

In the case that a state has more than one step, a subprocess in terms of BPMN can be generated from the state. In general, the generation of a subprocess follows the same scheme as the generation of BPMN process elements from an object (cf. Sect 4.2.1 - 4.2.5). Nevertheless, there are still differences that need to be considered when generating a subprocess with the proposed transforamtion model:

1. Activity Generation. In general, every step can be generated into an activity based on the Definition 2, 2.1 and 2.2, where the only difference is the naming of the respective activity. Depending on the data type of the respective step, the name of the generated activity from the step will either start with *Read* or *Write* followed by the attirbute instance name. The former case occurs, if the data type of the respective step is another related object. In such a case, the completion of the step implicitly demands, that another object instance is provided. Hence, the required data to complete the step has to be provided by the process itself and therefore can be read. On the contrary, the latter case occurs, if the data type of the respective step is not another related object (e.g., String, Boolean, Integer), hence, the completion of the step does not demand the provision of data from the process itself, but rather demands new data to be written externally from the process.

2. Data Object Generation.

TBD

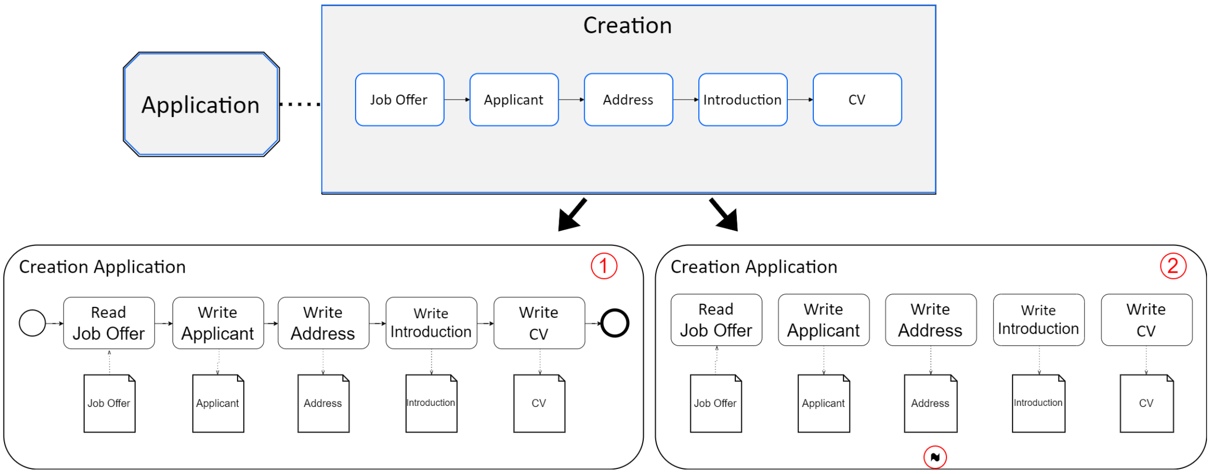

Abbildung 4.7: Initial example

B. Multi Instance Pool

text

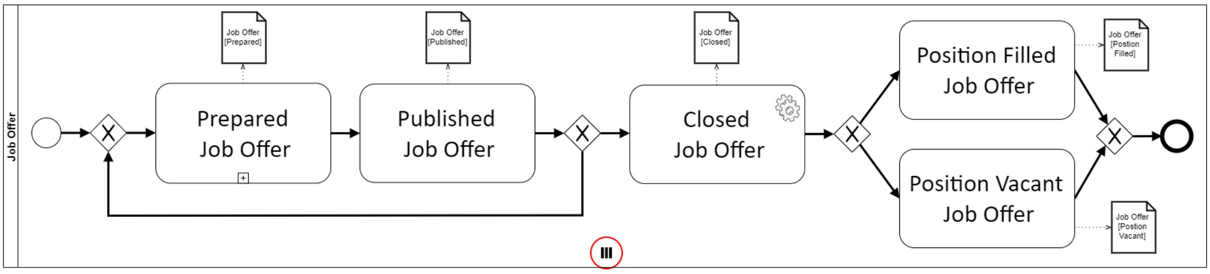

Abbildung 4.8: Initial example

C. Predicate Steps

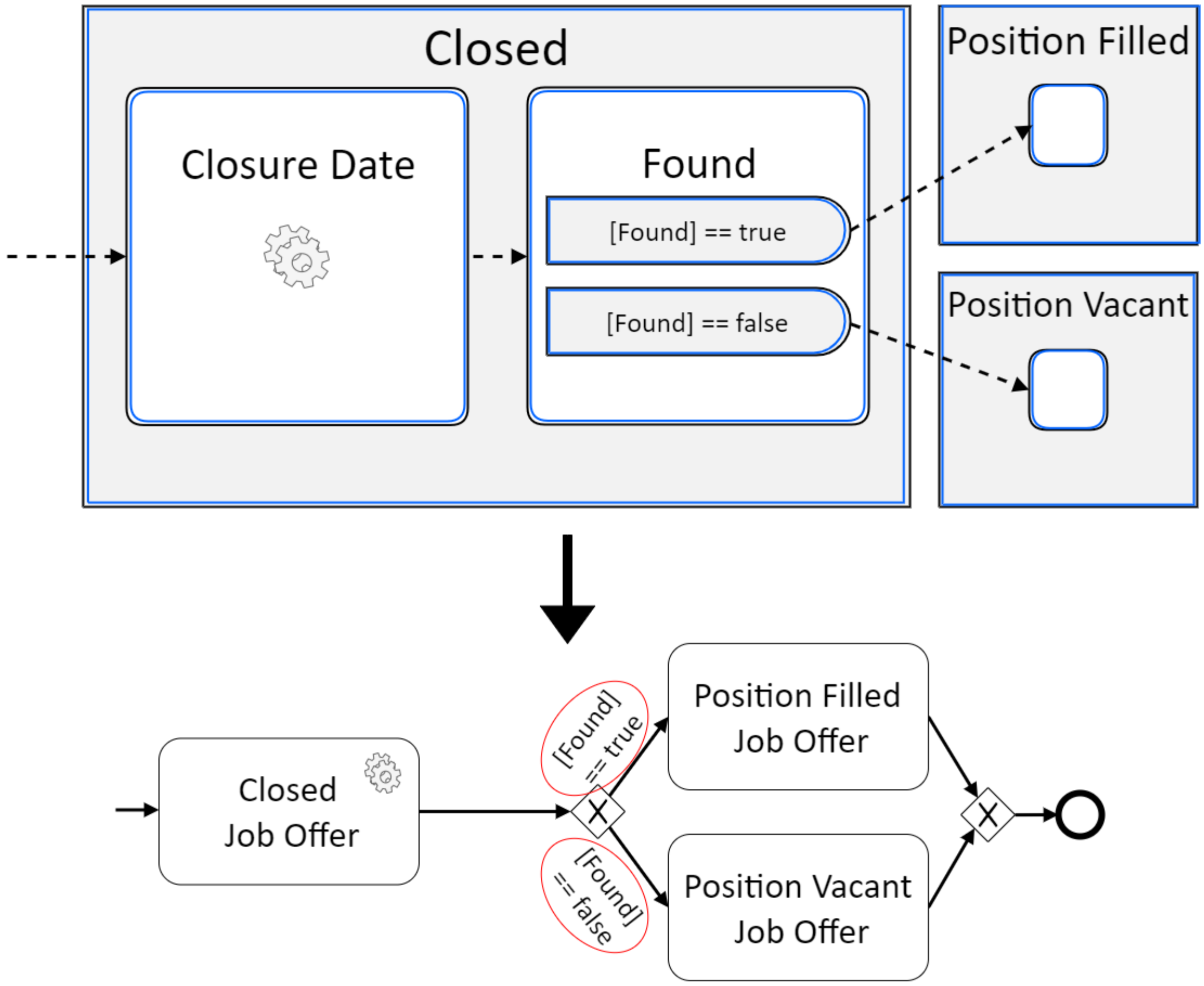

Abbildung 4.9: Initial example

4.3 Permission Integration

4.4 Relation Integration

4.5 Robotic Process Automation

4.6 Transformation Algorithm

5 Implementation

- Explain Algorithm in words and provide Pseudocode

6 Evaluation

6.1 Functional Requirements

6.2 Non-Functional Requirements

6.3 Limitations

7 Related Work

8 Conclusion

8.1 Contribution

8.2 Outlook

A Attachments

In diesem Anhang sind einige wichtige Quelltexte aufgeführt.

```java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

Literatur

[1] Jörg Knappen. *Schnell ans Ziel mit LATEX 2e*. 3., überarb. Aufl. München: Oldenbourg, 2009.

[2] Frank Mittelbach, Michel Goossens und Johannes Braams. *Der Latex-Begleiter*. 2., überarb. und erw. Aufl. ST - Scientific tools. München [u.a.]: Pearson Studium, 2005.

[3] Joachim Schlosser. *Wissenschaftliche Arbeiten schreiben mit LATEX : Leitfaden für Einsteiger*. 5., überarb. Aufl. Frechen: mitp, 2014.

[4] Thomas F. Sturm. *LATEX : Einführung in das Textsatzsystem*. 9., unveränd. Aufl. RRZN-Handbuch. Hannover [u.a.]: Regionales Rechenzentrum für Niedersachsen, RRZN, 2012.

[5] Herbert Voß. *LaTeX Referenz*. 2., überarb. u. erw. Aufl. Berlin: Lehmanns Media, 2010.

Name: Marko Pejic Matrikelnummer: 1027682

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den ....

Marko Pejic