
universität
**u**ulm

**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 2. Dezember 2022



Satz: PDF-LaTeX 2$_\varepsilon$

# 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

**Transformation Rule 1** (Step Type)**.** Let $\gamma$ = ($\phi$, $\sigma$, $\mathrm{T}_{in}$, $\mathrm{T}_{out}$, $\mathrm{P}$, $\lambda$) be a step. Then:

**StepType:** $\Gamma \mapsto$ ActivityStepTypes with:

$$\textbf{StepType}(\gamma) := \begin{cases} \text{Empty} & \text{if } \gamma.\phi = \bot, \\ \text{Computational} & \text{if } \gamma.\lambda \neq \bot, \\ \text{PredicateMacro} & \text{if } \gamma.\mathrm{P} \neq \bot, \\ \text{Predicate} & \text{if IsPredicate}(\gamma) = \text{true}, \\ \text{Task Step} & \text{else.} \end{cases}$$

**Transformation Rule 2** (Is Object)**.** Let $\Omega$ be the set of all objects within the process. Further, let $\phi$ be a reference to an attribute from $\Phi$ of an object $\omega \in \Omega$. Then:

$$\textbf{IsObject:}\ \Phi \mapsto \mathbb{B} \text{ with: } \textbf{IsObject}(\phi) = \begin{cases} \text{true} & \text{if } \exists \omega \in \Omega : \phi = \omega, \\ \text{false} & \text{else.} \end{cases}$$

**Transformation Rule 3** (Initialize Data Object)**.** Let $\alpha$ be an activity. Then:

**DataObjectInit:** $\Sigma \cup \Gamma \mapsto \mathrm{N}$ with:

- **DataObjectInit(**$\sigma$**)** := $(\mathrm{name})$ where
  - $\mathrm{name}$ = $\sigma.\omega.\mathrm{name}$ + $\sigma.\mathrm{name}$
- **DataObjectInit(**$\gamma$**)** := $(\mathrm{name})$ where
  - $\mathrm{name}$ = $\gamma.\phi$

**Transformation Rule 4** (Activity State Type)**.** Let $\alpha$ be an activity derived from a state $\sigma$ where $\Gamma_\sigma$ describes the set of steps belonging to $\sigma$. Then:

**ActivityType:** $\Gamma \mapsto$ ActivityStateTypes with:

$$\textbf{ActivityType}(\Gamma_\sigma) := \begin{cases} \text{Task} & \text{if } |\Gamma_\sigma| = 1 \wedge (\Gamma_\sigma.\gamma.\phi = \bot \vee \Gamma_\sigma.\gamma.\mathrm{P} \neq \bot), \\ \text{Sub-process} & \text{else.} \end{cases}$$

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.*

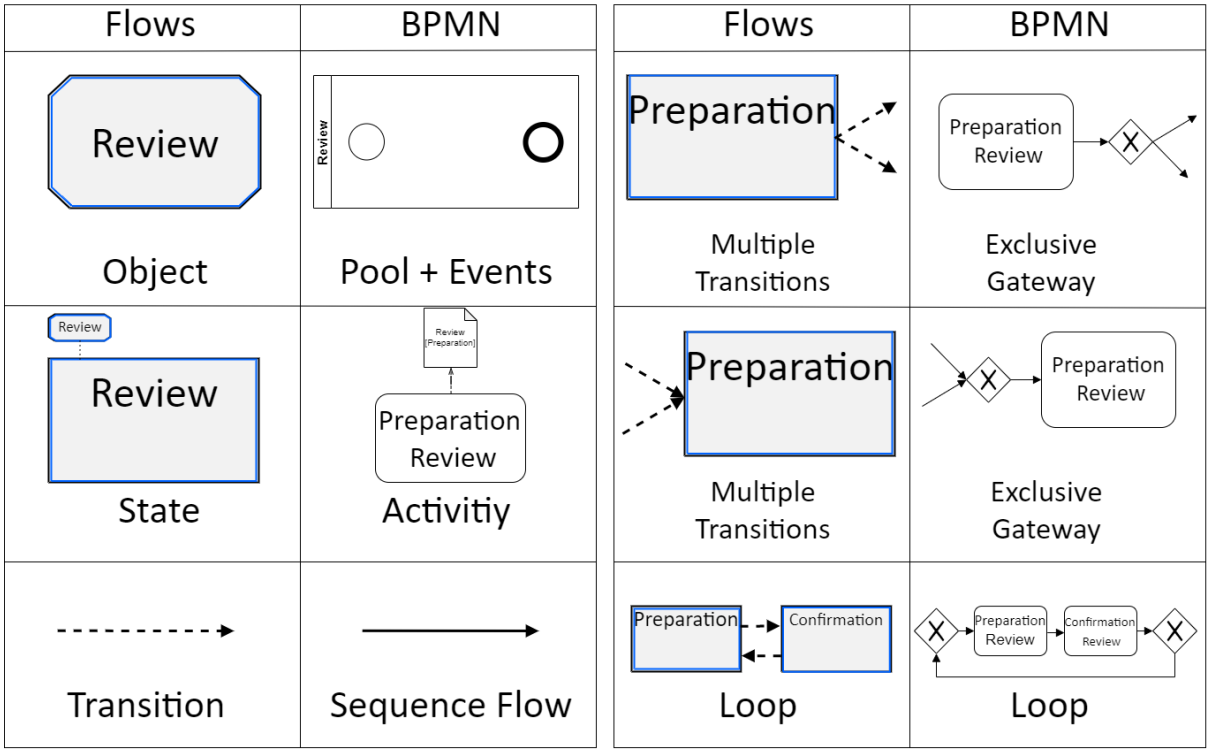


Abbildung 4.1: Initial example

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

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

## 4.2 Object Transformation

For the first step of the tranformation model, any given object $\Omega \ni \omega = (\mathrm{n}_\omega, \Phi, \Theta)$ is transformed in terms of BPMN. More specifically, transformation rules for the individual compenents within the lifecycle process $\Theta = (\Sigma, \Gamma, \mathrm{T}, \Psi)$ of $\omega$ are provided in this section. For the sake of completeness, other BPMN-related elements are included in the transformation, namley pools, events, gateways, and data objects.

**Transformation Rule 5** (Object Transformation)**.** *Let* $\omega = (\mathrm{n}_\omega, \Phi, \Theta)$ *be an object. Then: function* $\mathrm{OTA} : \Theta \rightarrow \delta$ *determines a 5-tuple of BPMN process elements* $\delta = (\mathrm{E}, \mathrm{A}, \mathrm{T}_\delta, \mathrm{K}, \mathrm{N})$*, derived from the lifecycle process* $\Theta.\omega$*.*

Note that, $\mathrm{OTA}$ is only concerned with the determination of a 5-tuple of BPMN process elements $\delta_\rho$, that, at the point of determination, belongs to an already predefined pool $\rho = (\omega_\rho, n_\rho, \delta_\rho)$.

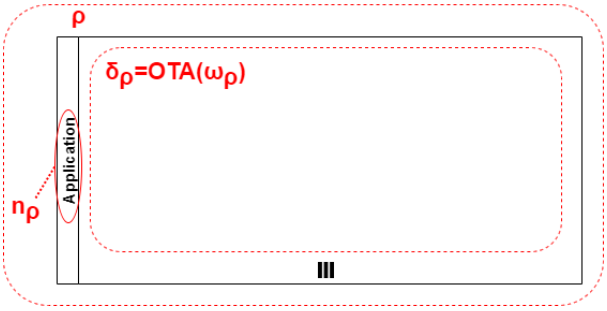


Abbildung 4.2: Initial example

As depicted in Figure TBD, transforming the object $\omega_{App.}$ requires a pool $\rho$, where $\rho.\omega_\rho = \omega_{App.}$ and, for its name, it holds $\rho.\mathrm{n}_\rho = \mathrm{n}_\omega$ = *Application*. Furthermore, the execution of a lifecycle usually refers to one instance of the respective object. In most cases, there will be multiple instances of an object, each of which has its own lifecycle to execute. In BPMN, the presence of multiple instances, or rather multiple participants, can be modeled by calling the corresponding pool a *multi instance* pool by drawing three vertical lines in the bottom center of the pool. For example, the multiplicity of $\rho_{App.}$ indicates that multiple applications can exist simultaneously and thus multiple processes involving $\omega_{App.}$ can exist independently.

Finally, as mentioned above, $\rho.\delta_\rho$ is still to be determined according to Algorithm 1.

**Algorithm 1** Object Transformation Algorithm (OTA)

**Require:** $\omega = (\mathrm{name}, \Phi, \Theta)$
  $\mathrm{E} \leftarrow \{\epsilon_{start}, \epsilon_{end}\}$
  **for each** $\sigma \in \Sigma$ **do**
    $\alpha \leftarrow$ StateTransformation($\sigma$)
    $\mathrm{A}$.add($\alpha$)
  **end for**
  **for each** $\alpha \in \mathrm{A}$ **do**
    **if** $\alpha.\iota$ = Sub-process **then**
      $\alpha.\delta.\mathrm{E} \leftarrow \{\epsilon_{start}, \epsilon_{end}\}$
      **for each** $\gamma \in \alpha.\Gamma_\sigma$ **do**
        $stepType \leftarrow$ StepType($\gamma$)
        **if** $stepType$ = Computational **then**
          $\widetilde{\alpha} \leftarrow$ ComputationalStepTransformation($\gamma$)
          $\alpha.\delta.\mathrm{A}$.add($\widetilde{\alpha}$)
        **else if** $stepType$ = Task **then**
          $\widetilde{\alpha} \leftarrow$ TaskStepTransformation($\gamma$)
          $\alpha.\delta.\mathrm{A}$.add($\widetilde{\alpha}$)
        **else if** $stepType$ = PredicateMacro **then**
          $\kappa \leftarrow$ PredicateMacroStepTransformation($\gamma$)
          **if** $\gamma = \alpha.\sigma.\gamma_{end}$ **then**
            $\mathrm{K}$.add($\kappa$)
          **else**
            $\alpha.\delta.\mathrm{K}$.add($\kappa$)
          **end if**
        **end if**
      **end for**
    **else if** $|\alpha.\Gamma_\sigma| = 1 \wedge$ StepType($\alpha.\Gamma_\sigma.\gamma$) = PredicateMacro **then**

```
        κ ← PredicateMacroStepTransformation(γ)
        K.add(κ)
    end if
end for
for each τω ∈ Θ.T do
    Tδ ← TransitionTransformation(τω)
    if τω ≠ ⊥ then
        if τω.ext = false then
            α̃ ← GetActivity(τω.γsource)
            α̃.δ.T.add(Tδ)
        else
            T.add(Tδ)
        end if
    end if
end for
αstart ← GetActivity(Θ.σstart)
τδ ← (E.εstart, αstart, ⊥)
T.add(τδ)
for each σend ∈ Θ.Σend do
    αend ← GetActivity(σend)
    τδ ← (αend, E.εend, ⊥)
    T.add(τδ)
end for
for each α ∈ A do
    if α.ι = Sub-process then
        αstart ← GetActivity(α.σ.γstart)
        τδ ← (α.δ.E.εstart, αstart, ⊥)
        α.δ.T.add(τδ)
        for each γend ∈ α.σ.Γend do
            αend ← GetActivity(γend)
            τδ ← (α̃, α.δ.E.εend, ⊥)
            α.δ.T.add(τδ)
        end for
    end if
end for
ComplementaryTransitionTransformation(K, Tδ)
```

### 4.2.1 State Transformation

In general, data must be read or written for a state to be executed within an object's lifecycle process, although neither may be necessary (e.g., if the state contains only an empty step). In any case, however, the execution of a state represents an activity of the user and ultimately drives the process flow. In terms of BPMN, such activities are modeled by *tasks* and *subprocesses* (cf. Def. TBD).

Consequently, depending on the latter, a set of BPMN elements is either initialized or defined as $\perp$ (cf. Def. TBD).

For the former, the corresponding BPMN elements are determined in the *Object Transformation* itself (cf. Algorithm TBD). Furthermore, since the execution of the lifecycle process of an object is data-driven, data objects are created each time a state is transformed into an activity in order to represent this integration of data into the BPMN process model as

well. For this purpose, a data object is created for each activity (cf. Def. TBD), where each data object should show in which state the respective object is after the activity has been executed. Finally, it must be determined for each activity which data objects are written by an activity, i.e. change the state of the object with its execution, and which data objects are read by an activity, i.e. require a certain state of an object or the previous execution of an activity for its execution. At this point in the transformation, however, data objects are only written when the respective state is transformed into an activity, hence the set of read data objects for each activity is not considered for now.

The combined execution of the functions described in Def. TBD - Def. TBD finally transform a state in terms of BPMN (cf. Def TBD).

**Transformation Rule 6** (State Transformation)**.** Let $\sigma$ = ($\mathrm{name}$, $\omega$, $\Gamma_\sigma$, $\mathrm{T}_\sigma$, $\Psi_\sigma$) be a state. Then:

**StateTransformation:** $\Sigma \mapsto \mathrm{A}$ with: **StateTransformation($\sigma$)** := ($\mathrm{name}$, $\sigma$, $\iota$, $\delta$, $\nu_{write}$, $\perp$) where

- $\mathrm{name}$ = $\sigma.\mathrm{name}$ + $\sigma.\omega.\mathrm{name}$
- $\iota$ = ActivityType($\Gamma_\sigma$)
- $\delta$ = $\begin{cases} (\mathrm{E}, \mathrm{A}, \mathrm{T}, \mathrm{K}, \mathrm{N}) & \text{if } \iota = sub-process, \\ \perp & \text{else.} \end{cases}$
- $\nu_{write}$ = DataObjectInit($\sigma$)

**Example TBD (State Transformation):**
Consider Fig. TBD. The *application* micro process type includes the state $\sigma_{Check}$. Then: $\sigma_{Check}$ has only one empty step; i.e., $|\Gamma_{Check}| = 1 \wedge \Gamma_{Check}.\gamma.\phi = \perp$. Hence, StateTransformation($\sigma_{Check}$) = ($\sigma_{Check}$, $Check\ Application$, $task$, $\perp$, $Application\ [\ Check\ ]$, $\perp$) (cf. Fig. TBD).

**Example TBD (State Transformation):**
Consider Fig. TBD. The *application* micro process type includes the state $\sigma_{Create}$. Then: $\sigma_{Create}$ has more than one step; i.e., $|\Gamma_{Create}| > 1$. Hence, StateTransformation($\sigma_{Create}$) = ($\sigma_{Create}$, $Create\ Application$, $subprocess$, $\delta$, $Application\ [\ Create\ ]$, $\perp$) (cf. Fig. TBD).

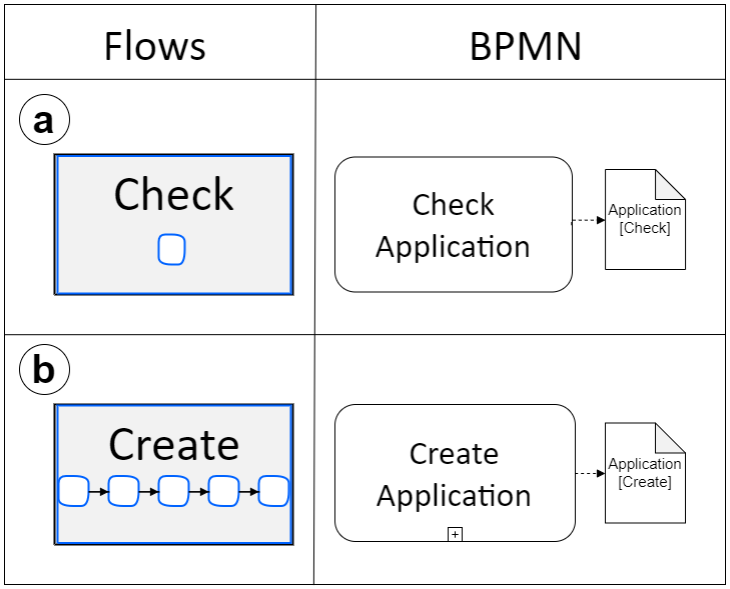


Abbildung 4.3: Initial example

### 4.2.2 Step Transformation

With the transformation of a state, its set of steps needs to be considered aswell. More specifically, for each sub-process within a BPMN process model, the set of steps of its corresponding state, needs to be transformed according to transformation rules aswell, where such transformation rule depends on the type of the step, which, for this purpose, is specified in Def. TBD. Ultimately, a step is transformed either into a *service task*, a *decision making procedure* or a *task*. In addition, every so generated (*service*) *task* generates a data object, for the same reason as described in the state transformation. However, a data object created in this way only describes what data the attribute of the respective step represents. As with the data objects created by states, data objects are either read or written by the corresponding (*service*) *task*, however, in contrary to the data objects created by states, the distinction between a read and written data object is made by the (*service*) *task* at the point of generation.

1. *Empty Step*

   If $\gamma$ is an empty step, it simultaneously holds, that $\gamma$ is the only step of the corresponding state $\gamma.\sigma$. Hence, the corresponding activity $\alpha$ that results from StateTransformation($\gamma.\sigma$) is considered a *task*. In this context, $\gamma$ is not taken into account in any way during the transformation, but for the sake of completeness, an example of such a case is illustrated in Fig. TBD (a).

2. *Computational Step*

   The transformation of a computational step $\gamma$ in terms of BPMN results in a *service task* (cf. Def. TBD).

**Transformation Rule 7** (Computational Step Transformation)**.** Let $\gamma$ = ($\phi$, $\sigma$, $\mathrm{T}_{in}$, $\mathrm{T}_{out}$, P, $\lambda$) be a step where $\gamma.\lambda$ = true. Then:

**ComputationalStepTransformation:** $\Gamma \mapsto \mathrm{A}$ with:
**ComputationalStepTransformation($\gamma$)** := ($\gamma$, name, service task, $\perp$, $\nu_{write}$, $\nu_{read}$) where

- name = $\begin{cases} \text{Read } + \gamma.\phi & \text{if IsObject}(\gamma.\phi) = \text{true}, \\ \text{Write } + \gamma.\phi & \text{else.} \end{cases}$
- $\widetilde{\nu}$ = DataObjectInit($\gamma$) with $\begin{cases} \nu_{read} = \widetilde{\nu} \wedge \nu_{write} = \perp & \text{if IsObject}(\gamma.\phi) = \text{true}, \\ \nu_{write} = \widetilde{\nu} \wedge \nu_{read} = \perp & \text{else.} \end{cases}$

**Example TBD (Computational Step Transformation):**
Consider Fig. TBD. The *application* micro process type includes the state $\sigma_{Sent}$, where $\sigma_{Sent}$ includes the *computational step* $\gamma$ = ($Sent\ Date$, $\sigma_{Sent}$, $\mathrm{T}_{in}$, $\mathrm{T}_{out}$, $\perp$, true), since $\gamma.\lambda$ = true. Furthermore, it holds IsObject($Sent\ Date$) = false. Then: $\alpha$ = StepTransformation($\gamma_{Sent\ Date}$) = ($\delta_{Sent\ App.}$, $Write\ Sent\ Date$, service task, $\perp$, $Sent\ Date$, $\perp$) (cf. Figure TBD (b)).

3. *Predicate Step*

   In the context of the transformation model, a special case occurs when $\gamma$ is considered as a predicate step, since no direct and complete transformation of $\gamma$ into the BPMN process model takes place. Rather, for the semantically correct translation of a predicate step, the step $\widetilde{\gamma}$ in which the $\gamma$ is located is considered. In fact, predicate steps are implicitly considered when a transformation to $\widetilde{\gamma}$ is performed.

4. *PredicateMacro*

   A lifecycle step $\gamma$ is specified as a *PredicateMacro Step* if it contains predicate step. For the transformation of such a *PredicateMacro Step*, a decision instance in terms of BPMN is applied, i.e., an exclusive gateway with different expressions is generated in order to represent the alternative process flow associated with $\gamma$ in the BPMN process model. For this purpose, as already mentioned above, each predicate step $\rho.\gamma \in \gamma.\mathrm{P}$ is implicitly taken into account, since these represent the expressions of the exclusive gateway. Def. TBD specifies the transformation of TBD.

**Transformation Rule 8** (PredicateMacro Transformation)**.** Let $\gamma$ = ($\phi$, $\sigma$, $\mathrm{T}_{in}$, $\mathrm{T}_{out}$, P, $\lambda$) be a step where $\gamma.\mathrm{P} \neq \perp$. Then:

**TBDTransformation:** $\Gamma \mapsto \mathrm{K}$ with: **TBDTransformation($\gamma$)** := ($\gamma$, P)

**Example TBD (PredicateMacro Step Transformation):**
Consider Fig. TBD. The *application* micro process type includes the state $\sigma_{Checked}$, where $\sigma_{Checked}$ includes the *PredicateMacro Step* $\gamma$ = ($Acceptance$, $\sigma_{Checked}$, $\mathrm{T}_{in}$, $\mathrm{T}_{out}$, P, false), since $\mathrm{P} = \{\rho_1, \rho_2\} \neq \bot$. Further, $\rho_1$ = ($\gamma_{PrdictStep1}$, $[Acceptance] == true$) and $\rho_2$ = ($\gamma_{PrdictStep2}$, $[Acceptance] == false$). Then: $\kappa_{split}$ = TBDStepTransformation($\gamma_{Application}$) = ($\gamma$, P) (cf. Figure TBD (c)).

5. *Task Step*

   If none of the above cases apply, $\gamma$ is transformed into a *task* (cf. Def. TBD).

**Transformation Rule 9** (Task Step Transformation)**.** Let $\gamma$ = ($\phi$, $\sigma$, $\mathrm{T}_{in}$, $\mathrm{T}_{out}$, P, $\lambda$) be a step with $\gamma.\lambda$ = false$\wedge\gamma.\mathrm{P} = \bot$. Then:

**TaskStepTransformation:** $\Gamma \mapsto \mathrm{A}$ with:
**TaskStepTransformation($\gamma$)** = ($\gamma$, name, task, $\bot$, $\nu_{write}$, $\nu_{read}$) where

- $\mathrm{name} = \begin{cases} \text{Read } + \gamma.\phi & \text{if IsObject}(\gamma.\phi) = \text{true}, \\ \text{Write } + \gamma.\phi & \text{else.} \end{cases}$
- $\widetilde{\nu}$ = DataObjectInit($\gamma$) with: $\begin{cases} \nu_{read} = \widetilde{\nu} \wedge \nu_{write} = \bot & \text{if IsObject}(\gamma.\phi) = \text{true}, \\ \nu_{read} = \bot \wedge \nu_{write} = \widetilde{\nu} & \text{else.} \end{cases}$

**Example TBD (Task Step Transformation):**
Consider Fig. TBD. The *application* micro process type includes the state $\sigma_{Create}$, which in turn, includes the *task step* $\gamma$ = ($Job\ Offer$, $\sigma_{Create}$, $\mathrm{T}_{in}$, $\mathrm{T}_{out}$, $\bot$, false), since $\gamma_{Job\ Offer}$ is neither a computational step, predicate step, nor an empty step; i.e., $\gamma.\lambda$ = false $\wedge$ $\gamma.\mathrm{P} = \bot \wedge \gamma.\phi \neq \bot$. Furthermore, it holds IsObject($Job\ Offer$) = true. Then: $\alpha$ = StepTransformation($\gamma_{Job\ Offer}$) = ($Create\ App.$, $Read\ Job\ Offer$, task, $\bot$, $\bot$, $Job\ Offer$) (cf. Figure TBD (d)).

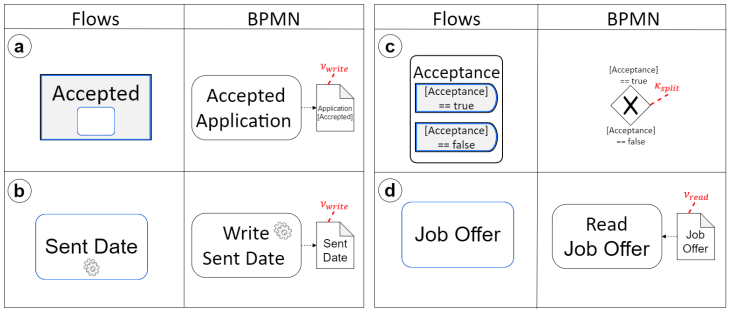


Abbildung 4.4: Initial example

Note that naming a task that is generated within the step transformation, i.e., transforming a task (computational) step $\gamma$ into a task $\alpha$, differs from naming a task, that is generated directly from a state within the state transformation. Thus, for the former, $\alpha.\mathrm{name}$ is determined by starting with either *Read* or *Write*, followed by the attribute instance $\gamma.\phi$. If latter refers to an object of the process (cf. Def. TBD), $\alpha.\mathrm{name}$ starts with *Read*, for the reason, that an instance of the data type corresponding to $\gamma.\phi$, must already exist at run-time, e.g., when executing $\gamma_{Job\ Offer} \in \Gamma_{Created}$, at least one instance of $\omega_{Job\ Offer}$ must already exist. Hence, $\gamma.\phi_{Job\ Offer}$ needs to be *read* within the process at run-time. In turn, $\alpha.\mathrm{name}$ starts with *Write*, if an instance of the data type corresponding to $\gamma.\phi$ must not necessarily exist at run-time, e.g., when executing $\gamma_{Address} \in \Gamma_{Created}$, the demanded data type can be provided at run-time. Hence, $\gamma.\phi_{Address}$ is *written* at run-time. In addition, the distinction between writing and reading a data type in the BPMN process model enables to implicitly show whether the execution of a task (state) depends on the execution (existence) of another task (object instance) (e.g., an application can only be created, if there is at least one job offer to apply to).

### 4.2.3 Transition Transformation

In general, any transition within an object's lifecycle process can be transformed, in terms of BPMN, into a sequence flow. Nevertheless, the semantically correct transformation depends on the properties of such a transition, as well as its source and its target. More precisely, of particular importance are the properties whether a transition is external or not, and whether a predicate step is part of the transition. Depending on the latter, different cases need to be considered for the transformation of a transition to be semantically correct. Algorithm TBD considers these cases and describes a general procedure how to transform any given transition in pseudo code.

**Algorithm 2** Transition Transformation

**Require:** $\tau_\omega = (\gamma_{source}, \gamma_{target}, \mathrm{ext})$
$\mathrm{T}_\delta \leftarrow$ **new**
$\mathrm{name} \leftarrow \perp$

**if** $\mathrm{ext} =$ false **then**
  **if** IsPredicateStep($\gamma_{source}$) $=$ true **then**
    $\upsilon_{source} \leftarrow$ GetGateway($\gamma_{source}$)
    $\mathrm{name} \leftarrow \gamma_{source}.\rho.\lambda$
  **else**
    $\upsilon_{source} \leftarrow$ GetActivity($\gamma_{source}$)
  **end if**
  **if** IsPredicateMacroStep($\gamma_{target}$) $=$ true **then**
    $\kappa \leftarrow$ GetGateway($\gamma_{target}$)
    **if** OnSecondGranularity($\kappa$) = true **then**
      $\upsilon_{target} \leftarrow \kappa$
    **end if**
  **else**
    $\upsilon_{target} \leftarrow$ GetActivity($\gamma_{target}$)
  **end if**
**else**
  **if** IsPredicateStep($\gamma_{source}$) $=$ true **then**
    $\upsilon_{source} \leftarrow$ GetGateway($\gamma_{source}$)
    $\mathrm{name} \leftarrow \gamma_{source}.\rho.\lambda$
    **if** GetPredicateMacroStep($\gamma_{source}$) $\notin$ Predecessors($\upsilon_{source}$) **then**
      $\tau_\delta \leftarrow$ (GetPredicateMacroStep($\gamma_{source}$), $\upsilon_{source}, \perp$)
      $\mathrm{T}_\delta$.add($\tau_\delta$)
    **end if**
  **else**
    $\upsilon_{source} \leftarrow$ GetActivity($\gamma_{source}.\sigma$)
  **end if**
  $\upsilon_{target} \leftarrow$ GetActivity($\gamma_{target}.\sigma$)
**end if**
**if** $\upsilon_{source} \neq \perp \wedge \upsilon_{target} \neq \perp$ **then**
  $\tau_\delta \leftarrow (\upsilon_{source}, \upsilon_{target}, \mathrm{name})$
  $\mathrm{T}_\delta$.add($\tau_\delta$)
**end if**
**return** $\mathrm{T}_\delta$

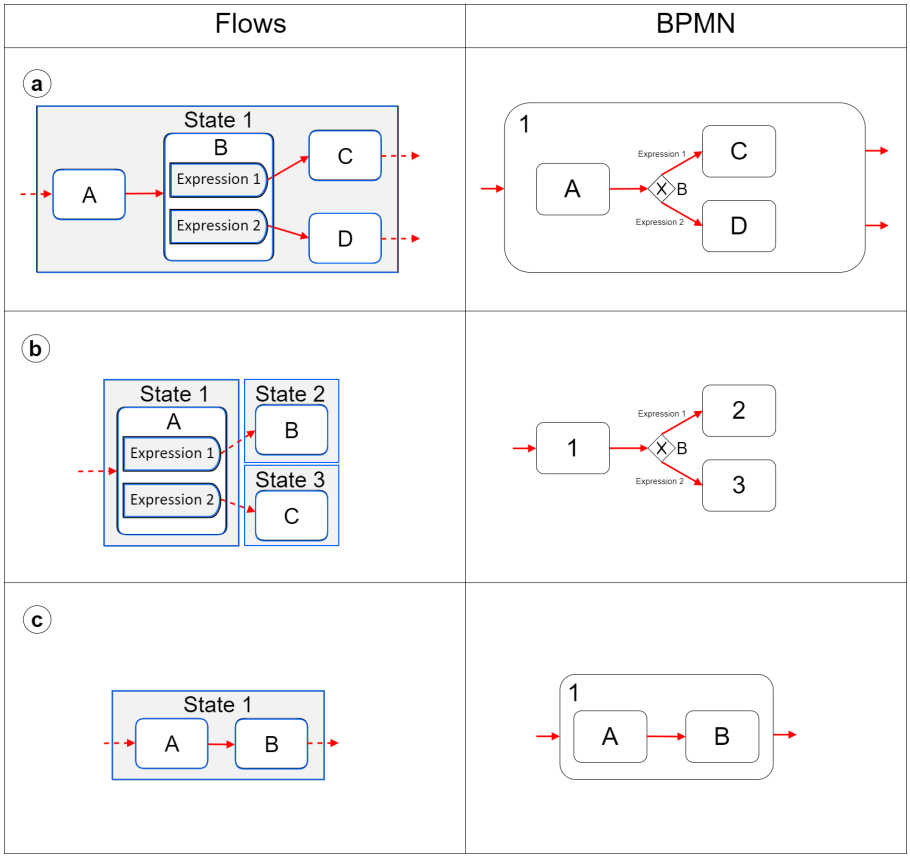


Abbildung 4.5: Initial example

As a result from Algorithm TBD, activities may have multiple incoming sequence flows. Consequently, for each such case, exclusive gateways are subsequently generated to group multiple incoming sequence flows of an activity. For this purpose, the sets of such sequence flows must be specified (cf. Def. TBD).

**Definition 9.1** (Multiple Target Sequence Flows).
Let $\mathrm{T}$ be a set of sequence flows. Further, let $\Upsilon$ be a set of BPMN elements where each $\upsilon \in \Upsilon$ serves as a target for a sequence flow. Then:

$$\mathrm{T}_{\mathbf{mltplTrgt}} := \forall \upsilon \in \Upsilon : \bigcup \{\tau_\delta | \tau_\delta \in \mathrm{T} \wedge \exists i, j \in \mathbb{N} : i \neq j \wedge \tau_\delta^i.\upsilon = \tau_\delta^j.\upsilon\} \subset \mathrm{T}.$$

Finally, Algorithm TBD completes the semantically correct transformation of transitions by describing the above metnioned generation of the exclusive gateways, as well as the adaptation of the associated sequence flows in pseudo code. More specifically, Algorithm TBD implicitly updates a given set of exclusive gateways $\mathrm{K}$ and $\mathrm{T}_\delta$ from a predefined 5-tuple of BPMN process elements $\delta$.

**Algorithm 3** Complementary Transition Transformation

**Require:** $\mathrm{K}, \mathrm{T}_{\delta}$
  **for each** $\mathrm{T}^{\upsilon}_{multipleTarget} \in \mathrm{T}_{multipleTarget}$ **do**
    $\kappa_{join} \leftarrow$ **new**
    $\tau_{\delta} \leftarrow (\kappa_{join}, \upsilon, \perp)$
    **for each** $\tau_{\delta} \in \mathrm{T}^{\upsilon}_{multipleTarget}$ **do**
      $\tau_{\delta}$.SetTarget($\kappa_{join}$)
    **end for**
    **if** IsSecondGranularity($\mathrm{T}^{\upsilon}_{multipleTarget}$) = false **then**
      $\widetilde{\alpha} \leftarrow \upsilon$
      $\widetilde{\alpha}.\delta.\mathrm{K}$.add($\kappa_{join}$)
      $\widetilde{\alpha}.\delta.\mathrm{T}$.add($\tau_{\delta}$)
    **else**
      K.add($\kappa_{join}$)
      T.add($\tau_{\delta}$)
    **end if**
  **end for**

### 4.2.4 Backwards Transition Transformation

**Algorithm 4** Backwards Transition Transformation

**Require:** $\mathrm{K}, \mathrm{T}_{\delta}$

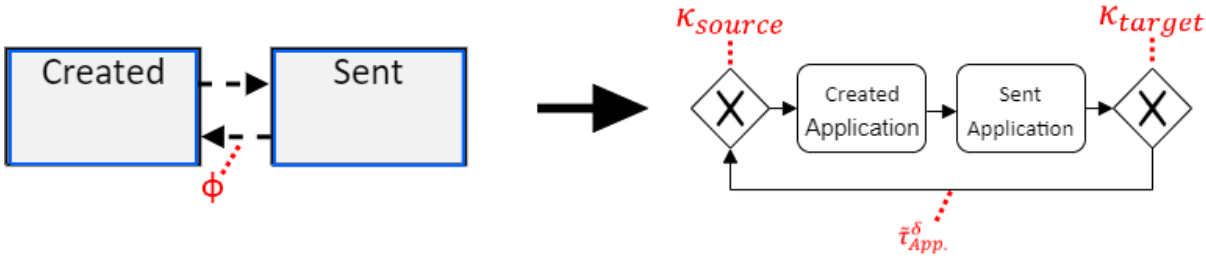


Abbildung 4.6: Initial example

## 4.3 Permission Integration

## 4.4 Relation Integration

## 4.5 Robotic Process Automation

## 4.6 Transformation Algorithm

**Algorithm 5** Step One (S1)

**Require:** $\Omega$
  $\mathrm{P} \leftarrow \perp$
  **for** $\omega \in \Omega$ **do**
    $\rho \leftarrow (\omega, \mathrm{n}_\omega, \perp)$
    $\mathrm{P}.add(\rho)$
  **end for**
  **for** $\rho \in \mathrm{P}$ **do**
    $\rho.\delta \leftarrow OTA(\rho.\omega)$
  **end for**
  **return** $\mathrm{P}$

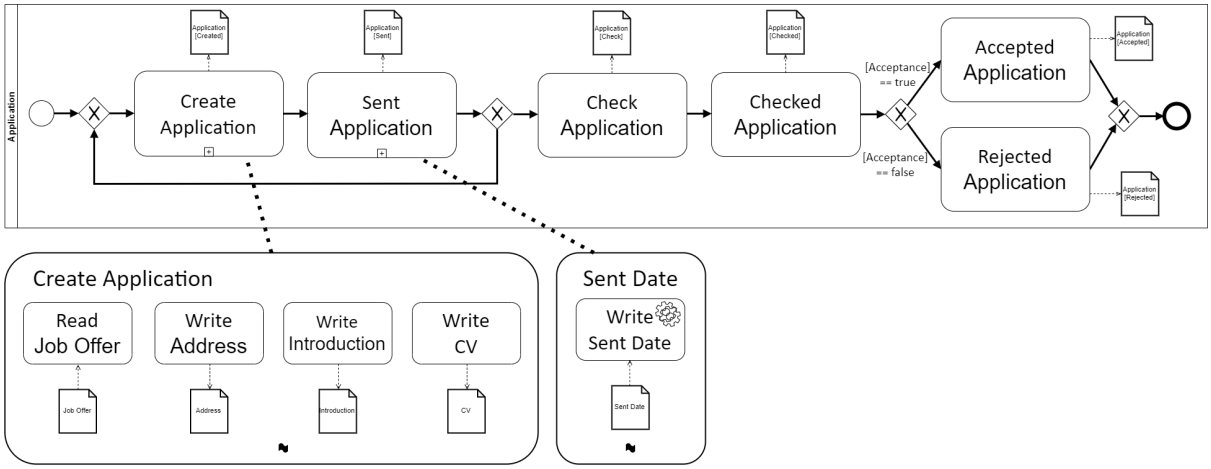


Abbildung 4.7: Initial example

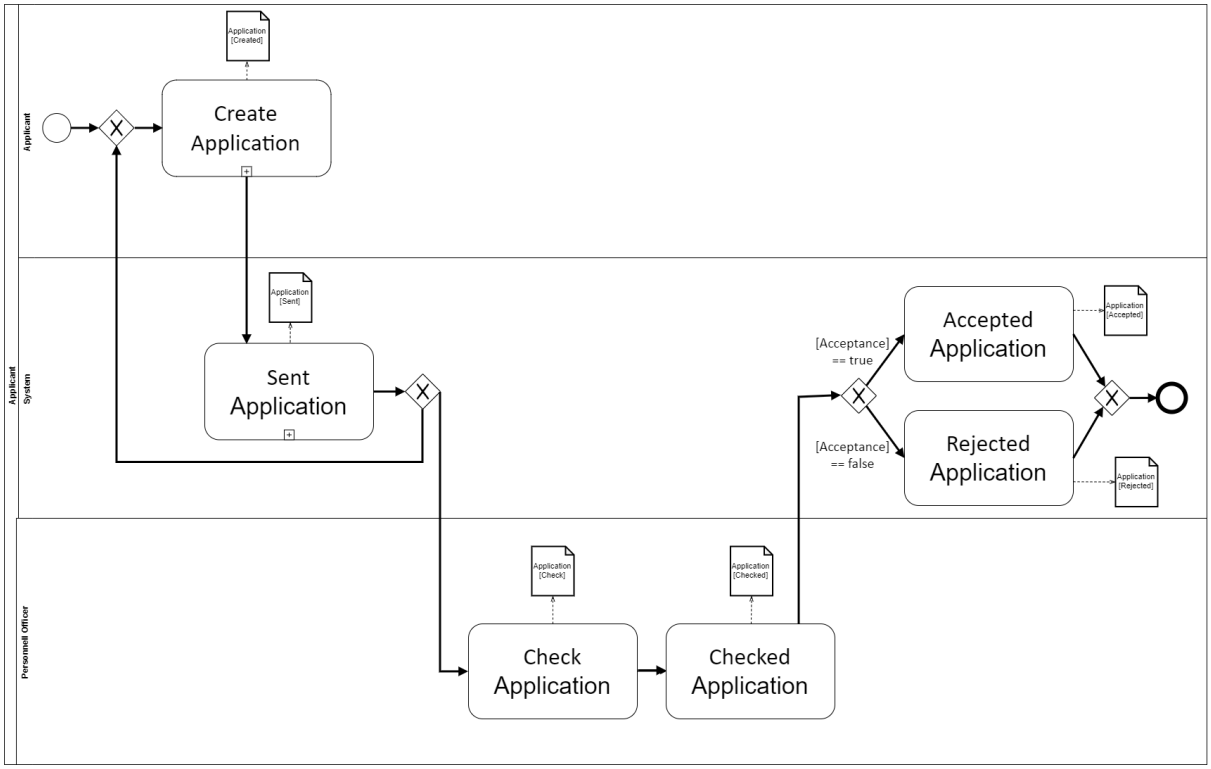


Abbildung 4.8: Initial example

# 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.

```
1 public class Hello {
2     public static void main(String[] args) {
3         System.out.println("Hello World");
4     }
5 }
```

# 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