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 6. 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

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.*

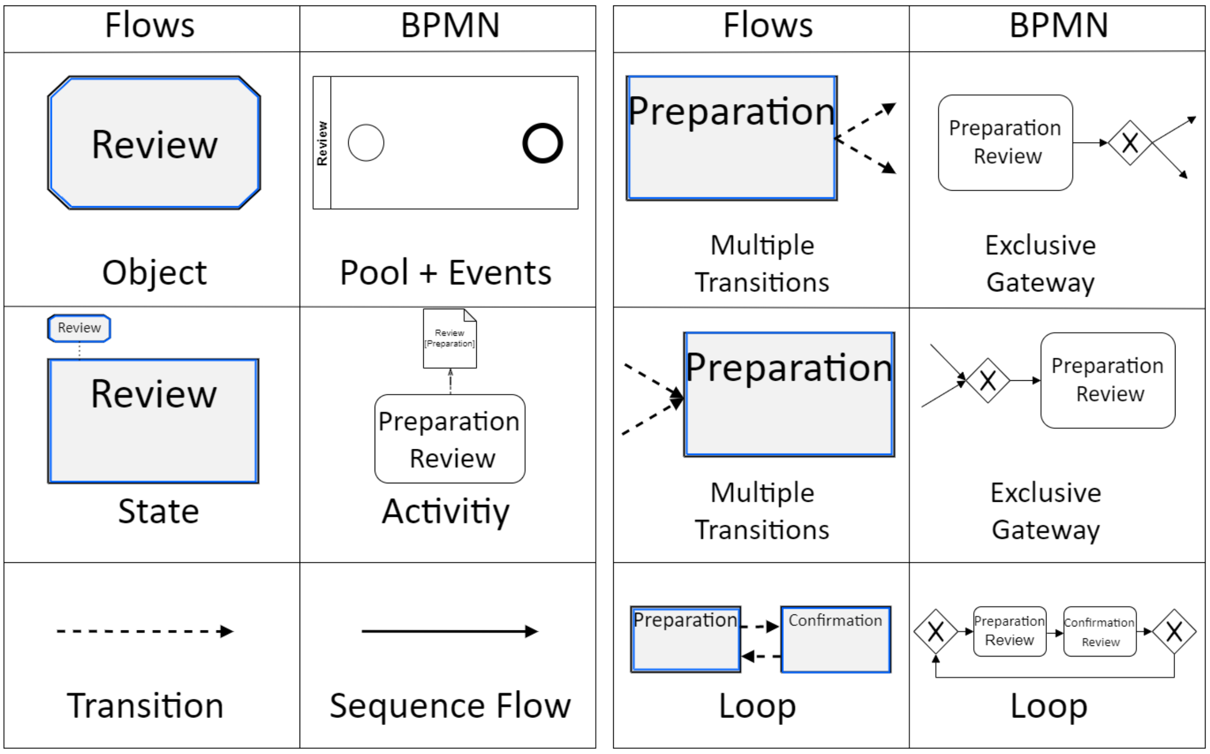

Abbildung 4.1: Initial example

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

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

4.2 Macro Process Model Transformation

Transforming a macro process model into a process model in terms of BPMN, requires the state-based view of an object's life cycle process. Based on this, an algorithm can be designed which, with the help of specified transformation rules, generates a process model that fits the requirements in terms of BPMN. More specificially, let $\omega = (\mathrm{n}, \Phi, \Theta)$ be an object. Given the state based view of the object's lifecycle process, i.e., Θ_{macro} = $(\omega, \Sigma, \mathrm{T}_{\Theta}^{ext}, \Psi, \sigma_{start}, \Sigma_{end}) \subset \Theta$, Algorithm TBD determines a pool $\rho = (\mathrm{n}, \delta)$, where δ contains further BPMN process modeling elements; i.e., events, activities, sequence flows, gateways and data objects. Note that in regard to a pool ρ it is important to mention, that the execution of an object's lifecycle process usually refers to one instance of the respective object. In most cases, however, there will be multiple instances of an object, each of which has its own lifecycle process 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 may exist simultaneously, each with its own independent process according to the lifecycle process of $\omega_{App.}$. The specification of Algorithm TBD is provided in pseudo code and, for illustration purposes, Fig. TBD provides an example of applying the algorithm to transform the macro process model of $\omega_{App.}$ in terms of BPMN.

Let Θ_{macro} = $(\omega, \Sigma, \mathrm{T}_{\Theta}^{ext}, \Psi, \sigma_{start}, \Sigma_{end})$ be the state based view of an object's lifecycle process. First of all, every component that shall be determined is initialized (line 1). After, the pool ρ is further declared where the name of ρ corresponds to the name of ω (line 2). Thereupon, a start and end event are generated (line 3) where the start event is associated with a data object (access = writing) that corresponds to the respective object (line 4-6), which represents the instantiation of the respective object with the start of its (lifecycle) process. Thereupon, every state $\sigma \in \Sigma$ is transformed into an actvitiy α (line 7-10). If the latter is considered a *sub-process*, its internal BPMN process elements need to be determined by considering the transformation of the micro process (cf. TBD). However, at this point, only the macro process is considered, hence, for now sub-processes are collapsed and their internal specification is considered as a black-box. Next, each transition $\tau_{\Theta} \in \mathrm{T}_{\Theta}^{ext}$ is transformed into a sequence flow (line 11-14). Furthermore, complementary sequence flows not covered by the transformation before, i.e, sequence flows including events, are added to the set of sequence flows within the BPMN process model (line 15-23). Furthermore, as a consequence of generating sequence flows as specified before, BPMN elements may have multiple incoming or outgoing sequence flows. Such cases are complementarily handeled by an auxiliary algorithm in line 24 (cf. Algorithm TBD). Last, each backwards transition $\psi \in \Psi$ is transformed (line 25-27) by generating additional sequence flows and exclusive gateways as specified in Algorithm TBD. In this context, the current set of sequence flows and exclusive gateways may be adapted aswell. To complete the algorithm, an output including ρ with the transformed elements stored in a 5-tuple δ is provided (line 28-30). Note that given a state type σ, the function GetBPMNElement(σ) provides the corresponding BPMNElement υ (i.e., α or ϵ) that, in general, already has been determined when executing the function. The remainder of this section describes the concrete transformation rules used in Algorithm TBD.

Algorithm 1 Macro Process Model Transformation

```
Require:  $\Theta_{macro}$ = $(\omega, \Sigma, \mathrm{T}_{\Theta}^{ext}, \Psi, \sigma_{start}, \Sigma_{end})$
Ensure:   $\rho$ = (n, $\delta$ = (
                $\mathrm{E}$:{$\epsilon_{start}$: Event, $\epsilon_{end}$: Event}
                $\mathrm{A}$:{$\alpha$: (n: String, $\iota$: ActivityType, $\delta$: 5-tuple, $\nu$: DataObject})
                $\mathrm{T}_{\delta}$:{$\tau_{\delta}$: ($\upsilon_{source}$: BPMNElement, $\upsilon_{target}$: BPMNElement, n: String})
                $\mathrm{K}$:{$\kappa$}
                $\mathrm{N}$:{$\nu$: (n: string, attributeType: AttributeType, access: {write, read})}
          ))
 1: $\rho, \mathrm{E}, \mathrm{A}, \mathrm{T}_{\delta}, \mathrm{K} \leftarrow$ new
 2: $\rho.\mathrm{n} \leftarrow \omega.\mathrm{n}$
 3: $\mathrm{E} \leftarrow \mathrm{E} \cup \{\epsilon_{start}, \epsilon_{end}\}$
 4: $\widetilde{\nu} \leftarrow$ ($\omega.\mathrm{n}$, Relation, write)
 5: $\epsilon_{start}.\nu \leftarrow \widetilde{\nu}$
 6: $\mathrm{N} \leftarrow \mathrm{N} \cup \{\nu\}$
 7: for each $\sigma \in \Theta.\Sigma$ do
 8:     $\alpha \leftarrow$ CreateMacroActivity($\sigma$)
 9:     $\mathrm{A} \leftarrow \mathrm{A} \cup \{\alpha\}$
```

```
end for
for each $\tau_\Theta \in \mathrm{T}_\Theta^{ext}$ do
    $\tau_\delta \leftarrow$ CreateMacroSequenceFlow($\tau_\omega$)
    $\mathrm{T}_\delta \leftarrow \mathrm{T}_\delta \cup \{\tau_\delta\}$
end for
$\alpha_{start} \leftarrow$ GetBPMNElement($\sigma_{start}$)
$\tau_\delta \leftarrow (\mathrm{E}.\epsilon_{start}, \alpha_{start}, \perp)$
$\mathrm{T}_\delta \leftarrow \mathrm{T}_\delta \cup \{\tau_\delta\}$
$\epsilon_{start} \leftarrow$ GetBPMNElement($\sigma_{start}$)
for each $\sigma_{end} \in \Sigma_{end}$ do
    $\alpha_{end} \leftarrow$ GetBPMNElement($\sigma_{end}$)
    $\tau_\delta \leftarrow (\alpha_{end}, \mathrm{E}.\epsilon_{end}, \perp)$
    $\mathrm{T}_\delta \leftarrow \mathrm{T}_\delta \cup \{\tau_\delta\}$
end for
ResolveMultipleSequenceFlows($\mathrm{T}_\delta$, K)
for each $\psi \in \Psi$ do
    CreateLoop($\psi$, $\mathrm{T}_\delta$, K)
end for
$\delta \leftarrow (\mathrm{E}, \mathrm{A}, \mathrm{T}_\delta, \mathrm{K})$
$\rho.\delta \leftarrow \delta$
return $\rho$
```

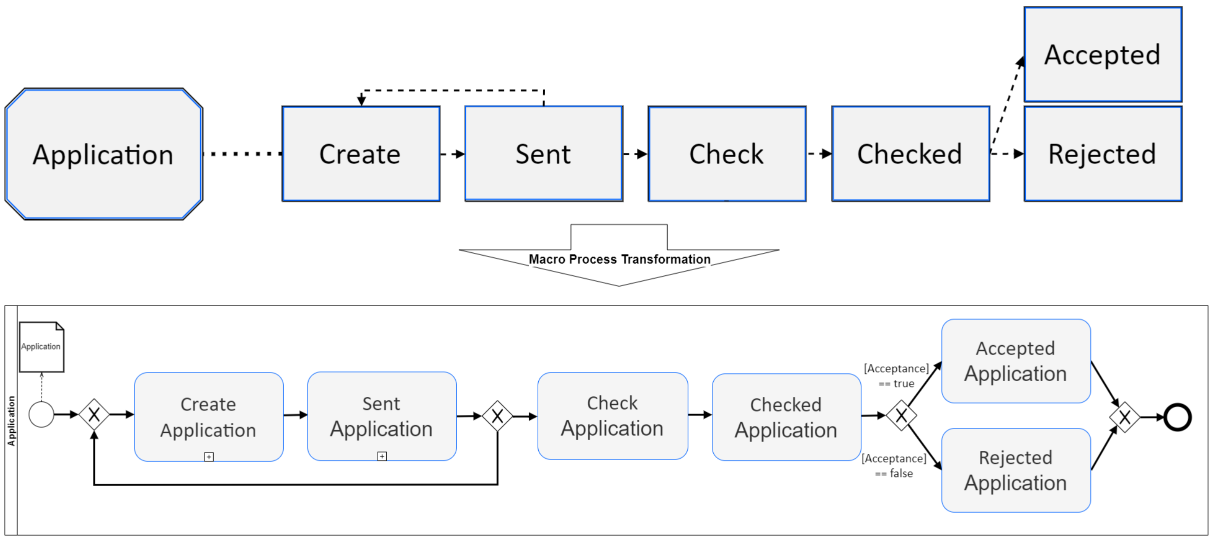

Abbildung 4.2: Initial example

4.2.1 State Type Transformation

In general, data must be read or written for a state type to be executed within an object's lifecycle process by executing its micro step types; although neither may be necessary (e.g., if the state contains only an empty micro step). In any case, however, a state type generates a form, where its form fields need to be filled with data (cf. Fig. TBD). For this purpose, PHILharmonicFlows provides form-based activites, which ultimately drives the process flow, while maintaining process contol. In order to represent such activities in terms of BPMN, the latter provides *tasks* and *sub-processes*. Hence, a transformation rules to transform a state type into a semantically correct activity in terms of BPMN, i.e., task or sub-process, must be specified (cf. TR TBD). Note that, in the case of the latter, the BPMN elements of the internal process are only initialized, whereas its concrete determination requires the transformation of the micro process (cf. TBD). At this point, however, it is sufficient to consider the internal process as a black-box. Example TBD & TBD close this chapter with an application of TR. TBD.

Transformation Rule 1 (State Type Transformation)**.** Let σ = $(\mathrm{n},\ \omega,\ \Gamma_\sigma,\ \mathrm{T}_\sigma,\ \Psi_\sigma,\ \gamma_{start},\ \Gamma_{end})$ be a state. Then:

CreateMacroActivity: $\Sigma \mapsto \mathrm{A}$ with:
CreateMacroActivity(σ) := $(\mathrm{n},\ \iota,\ \delta,\ \nu)$ where

- n = $\sigma.\mathrm{n}$ + $\sigma.\omega.\mathrm{n}$; i.e., state name followed by object name of the corresponding state.
- ι = $\begin{cases} \text{Task} & \text{if } |\Gamma_\sigma| = 1 \land (\Gamma_\sigma.\gamma.\phi = \bot \lor |\Gamma_\sigma.\gamma.\mathrm{P}|^{a} > 1), \\ \text{Sub-process} & \text{else.} \end{cases}$
- δ = $\begin{cases} \text{CreateSubProcess}(\sigma) = (\mathrm{E}, \mathrm{A}, \mathrm{T}, \mathrm{K}, \mathrm{N}) & \text{if } \iota = \text{Sub-process}, \\ \bot & \text{else.} \end{cases}$
- ν = $\bot$; TBD explanation.

[a] $\mathrm{P} = \bot \iff |\mathrm{P}| = 0$

Example TBD (State Type Transformation):
Consider Fig. TBD. The *application* micro process type includes the state type σ_{Check}. Then: σ_{Check} has only one empty micro step type; i.e., $|\Gamma_{Check}| = 1 \land \Gamma_{Check}.\gamma.\phi = \bot$. Hence, CreateMacroActivity(σ_{Check}) = (Check Application, $task$, $\bot$, $\bot$).

Example TBD (State Type Transformation):
Consider Fig. TBD. The *application* micro process type includes the state σ_{Create}. Then: σ_{Create} has more than one micro step type; i.e., $|\Gamma_{Create}| > 1$. Hence, CreateMacroActivity(σ_{Create}) = (Create Application, $sub-process$, δ, $\bot$).

4.2.2 Macro Transitions Type Transformation

In general, any transition within an object's lifecycle process can be transformed into a sequence flow. Nevertheless, the semantically correct transformation depends on the property, if the transition is external, as well as its source and target element. In regard to a macro process, all the transitions are external, hence, at this point a specification to specifically transform the latter must be provided. For this purpose, Algorithm TBD describes the transformation of external transitions into a sequence flow in pseudo code. Note that

Algorithm TBD uses the function GetBPMNElement(σ) that was already specified when describing Algorithm TBD. In addition, given a micro step type γ, the function StepType(γ) determines the kind of γ (cf. Def. TBD).

Algorithm 2 Create Macro Sequence Flow

```
Require: $\tau_\omega = (\gamma_{source}, \gamma_{target}, true)$
Ensure: $\mathrm{T}_\delta$
1:  $\mathrm{T}_\delta \leftarrow$ new
2:  $\mathrm{n} \leftarrow \perp$
3:  sourceStepType $\leftarrow$ StepType($\gamma_{source}$)
4:  if $sourceStepType(\gamma_{source}) = predicatestep$ then
5:      $\mathrm{n} \leftarrow \gamma_{source}.\rho.\lambda$
6:  end if
7:  $\upsilon_{source} \leftarrow$ GetBPMNElement($\gamma_{source}.\sigma$)
8:  $\upsilon_{target} \leftarrow$ GetBPMNElement($\gamma_{target}.\sigma$)
9:  $\tau_\delta \leftarrow (\upsilon_{source}, \upsilon_{target}, \mathrm{n})$
10: return $\tau_\delta$
```

Example TBD (Atomic Micro Step Type Transformation):
Consider Fig. TBD. The *application* micro process type includes the state σ_{Create}, which in turn, includes the atomic micro step type $\gamma_{Job\ Offer}$ = (Job Offer, σ_{Create}, T_{in}, T_{out}, $\perp$, false). Furthermore, it holds $\phi_{Job\ Offer}$ = Relation. Then: α = CreateMicroActivity($\gamma_{Job\ Offer}$) = (Read Job Offer, $task$, $\perp$, $\nu_{Job\ Offer}$) where $\nu_{Job\ Offer}$ = (Job Offer, Relation, read).

As a consequence from Algorithm TBD, activities and events may have multiple incoming or outgoing sequence flows. To resolve this, for each such case, exclusive gateways are subsequently generated to group multiple incoming and outgoing sequence flows of the corresponding activity or event. For this purpose, the set of such sequence flows must be specified (cf. Def. TBD & TBD).

Definition 1.1 (Multiple Incoming Sequence Flows).
Let $\delta = (\mathrm{E}, \mathrm{A}, \mathrm{T}_\delta, \mathrm{K}, \mathrm{N})$ be a 5-tuple of BPMN process elements. Further, let $\Upsilon = \mathrm{A} \cup \{\mathrm{E}.\epsilon_{end}\}$. Then:
$\mathrm{T}_\delta^{\mathbf{mltplTrgt}} := \bigcup_{\upsilon \in \Upsilon} \mathrm{T}_\upsilon^{mltplTrgt} \subset \mathrm{T}_\delta$ with:

$\mathrm{T}_\upsilon^{mltplTrgt} := \{\tau_\delta | \tau_\delta \in \mathrm{T}_\delta \wedge \tau_\delta.\upsilon_{target} = \upsilon \wedge \exists i, j \in \mathbb{N} : i \neq j \wedge \tau_\delta^i.\upsilon_{target} = \tau_\delta^j.\upsilon_{target}\}$.

Definition 1.2 (Multiple Outgoing Sequence Flows).
Let $\delta = (\mathrm{E}, \mathrm{A}, \mathrm{T}_\delta, \mathrm{K}, \mathrm{N})$ be a 5-tuple of BPMN process elements. Further, let $\Upsilon = \mathrm{A} \cup \{\mathrm{E}.\epsilon_{start}\}$. Then:
$\mathrm{T}_\delta^{\mathbf{mltplSrc}} := \bigcup_{\upsilon \in \Upsilon} \mathrm{T}_\upsilon^{mltplSrc} \subset \mathrm{T}_\delta$ with:

$$\mathrm{T}_{\upsilon}^{mltplSrc} := \{\tau_\delta | \tau_\delta \in \mathrm{T}_\delta \wedge \tau_\delta.\upsilon_{source} = \upsilon \wedge \exists i, j \in \mathbb{N} : i \neq j \wedge \tau_\delta^i.\upsilon_{source} = \tau_\delta^j.\upsilon_{source}\}.$$

Finally, Algorithm TBD completes the semantically correct transformation of transitions by describing the generation of exclusive gateways as mentioned above, 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 K and T_δ from a given 5-tuple of BPMN process elements δ and, therefore, does not provide any direct output. Note that given a sequence flow τ_δ and BPMNElement υ, the function SetTarget(τ_δ, υ) sets the target of τ_δ to υ. The function SetSource is defined analogously.

Algorithm 3 Resolve Multiple Sequence Flows

```
Require: $\mathrm{T}_\delta, \mathrm{K}$
 1: for each $\mathrm{T}_\upsilon^{mltplTrgt} \in \mathrm{T}_\delta^{mltplTrgt}$ do
 2:     $\kappa_{join} \leftarrow$ new
 3:     $\tau_\delta \leftarrow (\kappa_{join}, \upsilon, \perp)$
 4:     for each $\tau_\delta \in \mathrm{T}_\upsilon^{mltplTrgt}$ do
 5:         $\tau_\delta$.SetTarget($\kappa_{join}$)
 6:     end for
 7:     $\mathrm{K} \leftarrow \mathrm{K} \cup \kappa_{join}$
 8:     $\mathrm{T} \leftarrow \mathrm{T} \cup \tau_\delta$
 9: end for
10: for each $\mathrm{T}_\upsilon^{mltplSrc} \in \mathrm{T}_\delta^{mltplSrc}$ do
11:     $\kappa_{split} \leftarrow$ new
12:     $\tau_\delta \leftarrow (\upsilon, \kappa_{split}, \perp)$
13:     for each $\tau_\delta \in \mathrm{T}_\upsilon^{mltplSrc}$ do
14:         $\tau_\delta$.SetSource($\kappa_{split}$)
15:     end for
16:     $\mathrm{K} \leftarrow \mathrm{K} \cup \kappa_{split}$
17:     $\mathrm{T} \leftarrow \mathrm{T} \cup \tau_\delta$
18: end for
```

Example TBD (Atomic Micro Step Type Transformation):
Consider Fig. TBD. The *application* micro process type includes the state σ_{Create}, which in turn, includes the atomic micro step type $\gamma_{Job\ Offer}$ = (Job Offer, σ_{Create}, T_{in}, T_{out}, $\perp$, false). Furthermore, it holds $\phi_{Job\ Offer}$ = Relation. Then: α = CreateMicroActivity($\gamma_{Job\ Offer}$) = (Read Job Offer, $task$, $\perp$, $\nu_{Job\ Offer}$) where $\nu_{Job\ Offer}$ = (Job Offer, Relation, read).

Example TBD (Atomic Micro Step Type Transformation):
Consider Fig. TBD. The *application* micro process type includes the state σ_{Create}, which in turn, includes the atomic micro step type $\gamma_{Job\ Offer}$ = (Job Offer, σ_{Create}, T_{in}, T_{out}, $\perp$, false). Furthermore, it holds $\phi_{Job\ Offer}$ = Relation. Then: α = CreateMicroActivity($\gamma_{Job\ Offer}$) = (Read Job Offer, $task$, $\perp$, $\nu_{Job\ Offer}$) where $\nu_{Job\ Offer}$ = (Job Offer, Relation, read).

4.2.3 Backwards Transitions Transformation

Backwards transitions allow to reset the execution of a micro process by jumping back to a previous state. In turn, jumping back to a previous activity in an BPMN process model and reseting its execuction cannot be represented by a single process modeling element. For example, jumping back to a previous activity can be achieved by a loop, but the execution of the activity is not necesserialy reseted. Such a reset, however, can be represented by the data objects associated to the corresponding activities within an internal process, if the corresponding activity is considered a sub-process. The reason being, that the repeated execution of such an activity within the loop may write a new data object within the corresponding activity. Since an object can only be in one state at a time (cf. TBD), it appears as the execution of an activity has been reseted. Altogether, a backwards transition is transformed in terms of BPMN as specified in Algorithm TBD. Note that besides a backwards transitions ψ as input, Algorithm TBD only implicitly updates a given set of exclusive gateways K and T_δ from a given 5-tuple of BPMN process elements δ, since both is needed when transforming ψ and therefore does not provide any direct output. Further, the function Predecessor(α) determines the BPMNElement from which α is directly reachable, i.e., a BPMNElement υ for which it holds, that a sequence flow τ_δ = (υ, α, n) exists. In this regard, the function PredecessorSqncFlw(α) determines this corresponding sequence flow. The functions Successor and SuccessorSqncFlw are defined analogously. In addition, the functions SetTarget and SetSource as specified in chapter TBD are used in the algorithm.

Algorithm 4 Create Loop

```
Require: $\psi$ = ($\sigma_{source}$, $\sigma_{target}$), $\mathrm{T}_\delta$, $\mathrm{K}$
 1: $\alpha_{source} \leftarrow$ GetBPMNElement($\sigma_{source}$)
 2: $\alpha_{target} \leftarrow$ GetBPMNElement($\sigma_{target}$)
 3: $\upsilon_{pre} \leftarrow$ Predecessor($\alpha_{source}$)
 4: $\upsilon_{suc} \leftarrow$ Successor($\alpha_{target}$)
 5: $\tau_\delta^{pre} \leftarrow$ PredecessorSqncFlw($\alpha_{source}$)
 6: $\tau_\delta^{suc} \leftarrow$ SuccessorSqncFlw($\alpha_{target}$)
 7: if $\upsilon_{pre} \in \mathrm{K}$ then
 8:     $\kappa_{source} \leftarrow \upsilon_{pre}$
 9: else
10:     $\kappa_{source} \leftarrow$ new
11:     $\widetilde{\tau}_\delta \leftarrow (\kappa_{source}, \alpha_{source}, \perp)$
12:     $\tau_\delta^{pre}$.SetTarget($\kappa_{source}$)
13:     $\mathrm{K} \leftarrow \mathrm{K} \cup \kappa_{source}$
14:     $\mathrm{T}_\delta \leftarrow \mathrm{T}_\delta \cup \widetilde{\tau}_\delta$
15: end if
16: if $\upsilon_{suc} \in \mathrm{K}$ then
17:     $\kappa_{target} \leftarrow \upsilon_{suc}$
18: else
19:     $\kappa_{target} \leftarrow$ new
20:     $\widetilde{\tau}_\delta \leftarrow (\alpha_{target}, \kappa_{target}, \perp)$
21:     $\tau_\delta^{suc}$.SetSource($\kappa_{target}$)
22:     $\mathrm{K} \leftarrow \mathrm{K} \cup \kappa_{target}$
23:     $\mathrm{T}_\delta \leftarrow \mathrm{T}_\delta \cup \widetilde{\tau}_\delta$
24: end if
```

```
25: $\tau_\delta \leftarrow (\kappa_{target}, \kappa_{source}, \perp)$
26: $\mathrm{T}_\delta \leftarrow \mathrm{T}_\delta \cup \tau_\delta$
```

Example TBD (Atomic Micro Step Type Transformation):
Consider Fig. TBD. The *application* micro process type includes the state σ_{Create}, which in turn, includes the atomic micro step type $\gamma_{Job\ Offer}$ = (Job Offer, σ_{Create}, T_{in}, T_{out}, $\perp$, false). Furthermore, it holds $\phi_{Job\ Offer}$ = Relation. Then: α = CreateMicroActivity($\gamma_{Job\ Offer}$) = (Read Job Offer, $task$, $\perp$, $\nu_{Job\ Offer}$) where $\nu_{Job\ Offer}$ = (Job Offer, Relation, read).

4.3 Micro Process Model Transformation

In turn, transforming a micro process model into a process model in terms of BPMN, requires the micro process of an object's life cycle process. In contrast to the transformation of the macro process model, the transformation of the micro process model does not lead to a completely new BPMN process model. Rather, the latter extends an already transformed process model from the macro transformation to make its specification more precise. More specifically, with the help of the micro process model transformation, each sub-process from a BPMN process model derived from the macro process model transformation is specified, i.e., sub-processes are no longer consdiered as a black box, but as a white box. For this purpose, an algorithm must be specified that, with the help of specified transformation rules, determines the specification of such sub-processes in terms of BPMN. More specificially, let $\omega = (\mathrm{n}, \Phi, \Theta)$ be an object. Given the micro process model view of the object's lifecycle process, i.e., Θ_{micro} = $(\Sigma) \subset \Theta$, Algorithm TBD determines for each non-trivial state $\sigma \in \Sigma$ corresponding BPMN process modeling elements; i.e., events, activities, sequence flows, gateways and data objects. The specification of Algorithm TBD is provided in pseudo code and, for illustration purposes, Fig. TBD provides an example of applying the algorithm to transform the micro process model of $\omega_{App.}$ in terms of BPMN.

Let Θ_{micro} = (Σ) be the micro process model view of an object's lifecycle process. First of all, every component that shall be determined within the algorithm is initialized (line 1). Thereupon, a start and end event are generated (line 2). Next, every non-trivial state $\sigma \in \Sigma$ needs to be determined (line 3). For each such determined state σ, each step $\gamma \in \Gamma_\sigma$ is transformed depending on its kind (line 5-21). Then, each transition $\tau_\sigma \in \mathrm{T}_\sigma$ is transformed into a sequence flow (line 22-27). Furthermore, complementary sequence flows not covered by the transformation before, i.e, sequence flows including events, are added to the set of sequence flows within the BPMN process model (line 28-35). Furthermore, as in the macro process model transformation, BPMN elements may have multiple incoming or outgoing sequence flows. Such cases are complementarily handeled by the same auxiliary algorithm (cf. Algorithm TBD) as in the macro process model transformation in line 36. Last, an output including the transformed elements stored in a 5-tuple δ is provided (line 38-39). Note that given a state type γ, the function GetBPMNElement(γ) provides the corresponding BPMNElement υ (i.e., α, ϵ or κ) that, in general, already has been determined when executing the function. Further, given a micro step type γ, the function StepType(γ) determines the kind of γ (cf. Def. TBD). The remainder of this section describes the concrete transformation rules used in Algorithm TBD.

Algorithm 5 Micro Process Model Transformation

```
Require:  $\Theta_{micro}$ = $(\Sigma)$
Ensure:   $\delta$ = (
              $\mathrm{E}$:{$\epsilon_{start}$: Event, $\epsilon_{end}$: Event}
              $\mathrm{A}$:{$\alpha$: ($\mathrm{n}$: String, $\iota$: ActivityType, $\delta$: 5-tuple, $\nu$: DataObject})
              $\mathrm{T}_\delta$:{$\tau_\delta$: ($\upsilon_{source}$: BPMNElement, $\upsilon_{target}$: BPMNElement, $\mathrm{n}$: String})
              $\mathrm{K}$:{$\kappa$}
              $\mathrm{N}$:{$\nu$: ($\mathrm{n}$: string, $\mathrm{attributeType}$: AttributeType, $\mathrm{access}$: {$\mathrm{write}, \mathrm{read}$})}
          )
 1: $\mathrm{E}, \mathrm{A}, \mathrm{T}_\delta, \mathrm{K}, \mathrm{N} \leftarrow$ new
 2: $\mathrm{E} \leftarrow \mathrm{E} \cup \{\epsilon_{start}, \epsilon_{end}\}$
 3: for each $\sigma \in \{\sigma | \sigma \in \Sigma :!(|\Gamma_\sigma| = 1 \wedge (\Gamma_\sigma.\gamma.\phi = \perp \vee \Gamma_\sigma.\gamma.\mathrm{P} \neq \perp))\}$ do
 4:     $\alpha \leftarrow$ GetBPMNElement($\sigma$)
 5:     for each $\gamma \in \Gamma_\sigma$ do
 6:         $stepType \leftarrow$ StepType($\gamma$)
 7:         if $stepType$ = *value-specific micro step* then
 8:             if $|\gamma.\mathrm{P}| = 1$ then
 9:                 $\widetilde{\alpha} \leftarrow$ CreateMicroActivity($\gamma$)
10:                 $\mathrm{A} \leftarrow \mathrm{A} \cup \{\alpha\}$
11:                 $\mathrm{N} \leftarrow \mathrm{N} \cup \{\alpha.\nu\}$
12:             else if $|\gamma.\mathrm{P}| > 1 \wedge \gamma \notin \Gamma_{end}$ then
13:                 $\kappa \leftarrow$ new
14:                 $\mathrm{K} \leftarrow \mathrm{K} \cup \{\kappa\}$
15:             end if
16:         else
17:             $\widetilde{\alpha} \leftarrow$ CreateMicroActivity($\gamma$)
18:             $\mathrm{A} \leftarrow \mathrm{A} \cup \{\alpha\}$
19:             $\mathrm{N} \leftarrow \mathrm{N} \cup \{\alpha.\nu\}$
20:         end if
21:     end for
22:     for each $\tau_\Theta \in \mathrm{T}_\Sigma$ do
23:         $\tau_\delta \leftarrow$ CreateMicroSequenceFlow($\tau_\Theta$)
24:         if $\tau_\delta \neq \perp$ then
25:             $\mathrm{T}_\delta \leftarrow \mathrm{T}_\delta \cup \{\tau_\delta\}$
26:         end if
27:     end for
28:     $\upsilon_{start} \leftarrow$ GetBPMNElement($\gamma_{start}$)
29:     $\tau_\delta \leftarrow (\mathrm{E}.\epsilon_{start}, \upsilon_{start}, \perp)$
30:     $\mathrm{T}_\delta \leftarrow \mathrm{T}_\delta \cup \{\tau_\delta\}$
31:     for each $\gamma_{end} \in \Gamma_{end}$ do
32:         $\upsilon_{end} \leftarrow$ GetBPMNElement($\gamma_{end}$)
33:         $\tau_\delta \leftarrow (\upsilon_{end}, \mathrm{E}.\epsilon_{end}, \perp)$
34:         $\mathrm{T}_\delta \leftarrow \mathrm{T}_\delta \cup \{\tau_\delta\}$
35:     end for
36:     ResolveMultipleSequenceFlows($\mathrm{T}_\delta, \mathrm{K}$)
37: end for
38: $\delta \leftarrow (\mathrm{E}, \mathrm{A}, \mathrm{T}_\delta, \mathrm{K}, \mathrm{N})$
39: return $\delta$
```

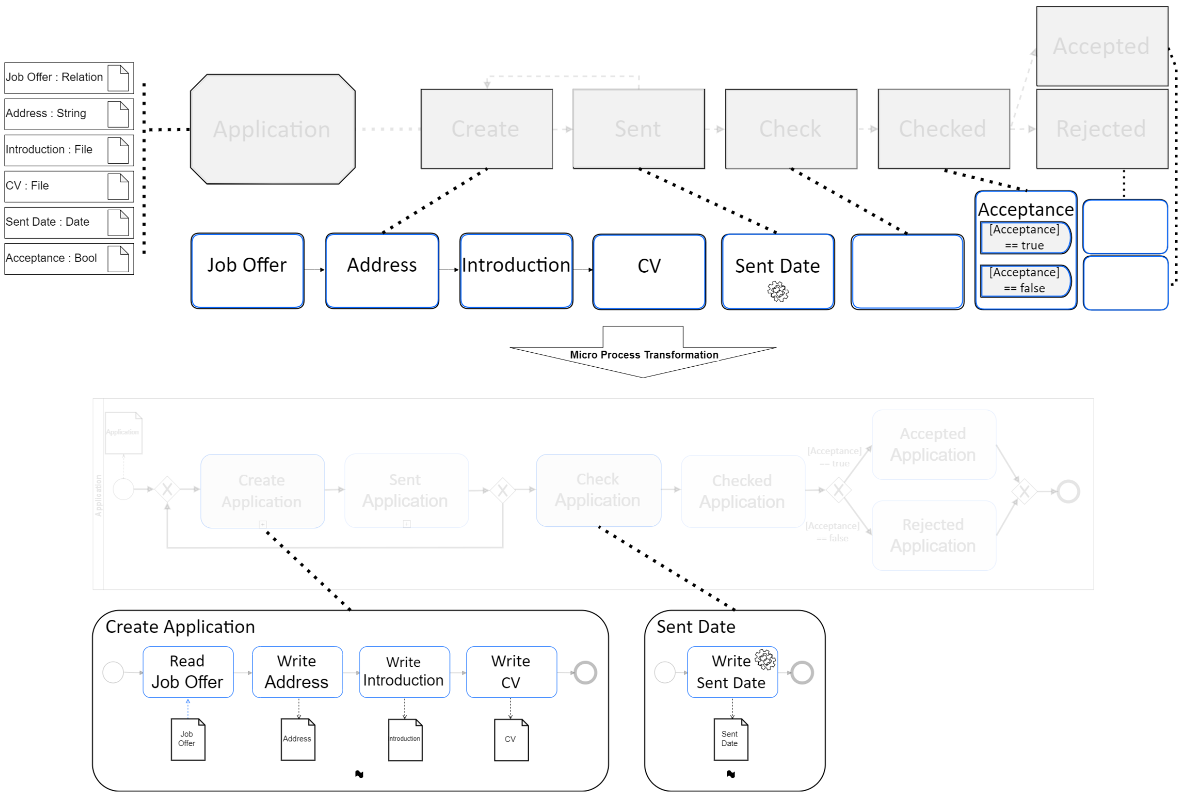

Abbildung 4.3: Initial example

4.3.1 Micro Step Type Transformation

TBD

As mentioned in the state type transformations. In other words, the execution of a micro step type is what makes the process data-driven in the first place, since data is provided to the process. The goal, when transforming a micro step type into its BPMN equivalent, therefore is, to make this data-driven approach apparent in BPMN. For this purpose, data objects are used, more specifically, for every applied transformation rule onto a micro step type (cf. TR TBD), a data object is generated. These transformation rules are described as follows. Note that given a micro step type γ, the function StepType(γ) determines the kind of γ. In the following, for the sake of brevity, micro step types will be referred to as steps.

Transformation Rule 2 (Step Type)**.** Let γ = (ϕ, σ, T_{in}, T_{out}, P, λ) be a micro step type. Then:

StepType: $\Gamma \mapsto$ StepTypes with:

$$\textbf{StepType}(\gamma) := \begin{cases} \text{Empty} & \text{if } \gamma.\phi = \perp, \\ \text{Predicate} & \text{if GetValueSpecificStep}(\gamma) \neq \perp, \\ \text{Value-Specific} & \text{if } \gamma.\mathrm{P} \neq \perp, \\ \text{Computation} & \text{if } \gamma.\lambda \neq \perp, \\ \text{Atomic} & \text{else.} \end{cases}$$

A. Empty Micro Step Type

If γ is considered an *empty step* , it simultaneously holds, that γ is the only step of its corresponding state type $\gamma.\sigma$ (cf. TBD). Hence, the corresponding activity α that results from CreateMacroActivity($\gamma.\sigma$) is considered a *task*. In this context, γ does not require any specific transformation rule and is practically skipped when in the context of the step transformation. For instance, Example TBD illustrates the transformation of a state type inluding an empty step.

B. Predicate Micro Step Type

In the context of the micro step type transformation, a special case occurs when γ is considered a *predicate step*, since no direct and complete transformation of γ into the BPMN process model takes place. Rather, only the expressions provided by predicate steps are of interest and implicitly considered when generating sequence flows (cf. TBD). In addition, predicate steps are of interest, because their existence indicates a *value-specific step*. As mentioned, predicate steps are implicitly considered when generating sequence flows, hence, no explicit transformation rule is provided in this chapter.

C. Value-Specific Micro Step Type

As mentioned above, transforming a *value-specific step* $\widetilde{\gamma}$ requires the consideration of its predicate steps. On the one hand, if $\widetilde{\gamma}$ contains at least two predicate steps, it is transformed into an *exclusive gateway* in terms of BPMN (cf. Example TBD) to control the convergence and divergence of sequence flows in the BPMN process that arises from a value-specific micro step type. On the other hand, if $\widetilde{\gamma}$ only contains one predicate step, $\widetilde{\gamma}$ is transformed into a task (cf. Example TBD), according to TR. TBD. Algorithm TBD distincts these cases directly. Since only an exclusive gateway needs to be generated for the first case and an already defined transformation rule is used for the second case, the transformation of a value-specific step does not require the specification of a new transformation rule.

Example TBD (TBD):
Consider Fig. TBD. The *application* micro process type includes the state $\sigma_{Checked}$ where $\sigma_{Acceptance}$ includes the *value-specific step* $\gamma_{Acceptance}$, since $\gamma_{Acc.}.\mathrm{P} \neq \perp$. Furthermore, it holds $\gamma_{Acc.}.\mathrm{P} = \{\rho_1, \rho 2\}$ where ρ_1 = (γ, Acceptance == [true]) and ρ_2 =

(γ, Acceptance == [false]). Then: An exclusive gateway κ is generated.

Example TBD (TBD):
Consider Fig. 1 in TBDCITEUIB201606. The *review* micro process type includes the state $\sigma_{Reject\ Proposed}$, where $\sigma_{Reject\ Proposed}$ includes the *value-specific step* $\gamma_{Finished}$, since $|\gamma_{Finished}.\mathrm{P}| > 1$. But, $\gamma_{Finished.}.\mathrm{P}$={($\gamma$, Finished == [true])}. Hence: CreateMicroActivity($\gamma_{Finished}$) = (Write Finished, $task$, $\perp$, $\nu_{Finished}$).

D. Atomic Micro Step Type and Computation Micro Step Type

If none of the above cases apply, γ is considered either a *atomic step* or a *computation step*. Since both cases are identical in their specification (except for their activity type), the same transformation rule TR TBD can be applied to both cases as follows. Example TBD illustrates the application of TR TBD onto an atomic step, whereas Example TBD illustrates applying TR. TBD onto a computation step.

Transformation Rule 3 (Create Micro Activity)**.** Let γ = (ϕ, σ, T_{in}, T_{out}, P, λ) be a lifecycle step with $\phi \neq \perp \wedge \mathrm{P} = \perp$; i.e., γ neither represents an empty micro step nor a predicate step. Then:

CreateMicroActivity: $\Gamma \mapsto \mathrm{A}$ with:
CreateMicroActivity(γ**)** = (n, ι, δ, ν) where

– $\mathrm{n} = \begin{cases} \text{Read} + \phi.\mathrm{n} & \text{if } \phi.\mathrm{attributeType} = \text{Relation,} \\ \text{Write} + \phi.\mathrm{n} & \text{else.} \end{cases}$

– $\iota = \begin{cases} \text{service task} & \lambda = \text{true,} \\ \text{task} & \text{else.} \end{cases}$

– δ = $\perp$; i.e., the respective activity does not contain an internal process to describe its activity.

– ν = ($\phi.\mathrm{n}, \phi.\mathrm{attributeType}$, access) with:

$$\text{access} = \begin{cases} \mathrm{read} & \text{if } \phi.\mathrm{attributeType} = \text{Relation,} \\ \mathrm{write} & \text{else.} \end{cases}$$

Example TBD (Atomic Micro Step Type Transformation):
Consider Fig. TBD. The *application* micro process type includes the state σ_{Create}, which in turn, includes the atomic micro step type $\gamma_{Job\ Offer}$ = (Job Offer, σ_{Create}, T_{in}, T_{out}, $\perp$, false). Furthermore, it holds $\phi_{Job\ Offer}$ = Relation. Then:

α = CreateMicroActivity($\gamma_{Job\ Offer}$) = (Read Job Offer, $task$, $\perp$, $\nu_{Job\ Offer}$) where $\nu_{Job\ Offer}$ = (Job Offer, Relation, read).

Example TBD (Computation Micro Step Type Transformation):
Consider Fig. TBD. The *application* micro process type includes the state σ_{Sent}, where σ_{Sent} includes the *computation micro step type* $\gamma_{Sent\ Date}$ = (Sent Date, σ_{Sent}, T_{in}, T_{out}, $\perp$, true), since λ = true. Furthermore, it holds $\phi_{Sent\ Date}$ = false. Then: CreateMicroActivity($\gamma_{Sent\ Date}$) = (Write Sent Date, $service\ task$, $\perp$, $\nu_{Sent\ Date}$) where $\nu_{Sent\ date}$ = (Sent Date, Date, write).

Note that naming a (service) task that is generated within the micro step type transformation, i.e., transforming an (computation) atomic step γ into a (service) task α (cf. Example TBD and TBD), differs from naming a task, that is generated directly from a state type. Thus, for the former, $\alpha.\mathrm{n}$ is determined by starting with either *Read* or *Write*, followed by the attribute instance ϕ demanded by γ. If latter relates to an object type within the process, i.e., $\phi.attributeType = Relation$, $\alpha.\mathrm{n}$ starts with *Read*, because an object instance of the attribute relating to γ must already exist at run-time. For example, when executing $\gamma_{Job\ Offer} \in \Gamma_{Created}$, at least one instance of $\omega_{Job\ Offer}$ must already exist, since $\phi_{Job\ Offer}.attributeType = Relation$. Hence, $\omega_{Job\ Offer}$ needs to be *read* within the process when executing $\gamma_{Job\ Offer}$ at run-time. Analogously, the value of access from the corresponding data object $\nu_{Job\ Offer}$ is set to read; i.e.; a reading association between $\alpha_{Read\ Job\ Offer}$ and $\nu_{Job\ Offer}$ is drawn. In turn, $\alpha.\mathrm{n}$ starts with *Write*, if an object instance of the attribute relating to γ must not necessarily exist at run-time, i.e., $\phi.attributeType \neq Relation$. As an example, when executing $\gamma_{Address} \in \Gamma_{Created}$, the attribute $\phi_{Address}$ can be provided at run-time, since it does not relate to an object type ($\phi_{Address}.attributeType = String \neq Relation$). Hence, $\phi_{Address}$ is *written* at run-time when executing $\gamma_{Address}$. Analogously, the value of access from the corresponding data object $\nu_{Address}$ is set to write; i.e.; a writing association between $\nu_{Address}$ and $\alpha_{Write\ Address}$ is drawn. In addition, the distinction between writing and reading an attribute in the BPMN process model implicitly shows whether the execution of an activity depends on the execution of another activity, potientially from another pool.

4.3.2 Micro Transitions Type Transformation

As in the macro transitions type transformation (cf. TBD), any transition within an object's lifecycle process can generally be transformed into a sequence flow. However, in regard to a micro process, all transitions are non-external, hence, this chapter provides a specification to transform these kind of transitions in sequence flows. For this purpose, Algorithm TBD describes the transformation of non-external transitions into a sequence flow in pseudo code. Note that Algorithm TBD uses the function GetBPMNElement(γ) that was already specified when describing Algorithm TBD. Further, the function StepType(γ) is used as specified in Def. TBD.

Algorithm 6 Create Micro Sequence Flow

```
Require: $\tau_\omega$ = $(\gamma_{source}, \gamma_{target}, false)$
Ensure: $\tau_\delta$
 1: $\tau_\delta \leftarrow$ new
 2: $\mathrm{n} \leftarrow \perp$
 3: sourceStepType $\leftarrow$ StepType($\gamma_{source}$)
 4: if $sourceStepType(\gamma_{source}) = predicatestep$ then
 5:     $\mathrm{n} \leftarrow \gamma_{source}.\rho.\lambda$
 6: end if
 7: if $sourceStepType(\gamma_{target}) = value - specificmicrostep \wedge \gamma_{target} = \gamma_{target}.\sigma.\gamma_{end}$ then
 8:     end algorithm
 9: end if
10: $\upsilon_{source} \leftarrow$ GetBPMNElement($\gamma_{source}$)
11: $\upsilon_{target} \leftarrow$ GetBPMNElement($\gamma_{target}$)
12: $\tau_\delta \leftarrow (\upsilon_{source}, \upsilon_{target}, \mathrm{n})$
13: return $\tau_\delta$
```

Similarly to the macro transitions type transformation, as a consequence of Algorithm TBD, activities and events may have multiple incoming or outgoing sequence flows. For this purpose, Def. TBD, Def. TBD and Algorithm TBD as specified in chapter TBD can be used by the micro transitions type transformation aswell to resolve this. Hence, no further transformation rules are provided.

4.4 Permission Integration

4.5 Coordination Process Transformation

4.6 Robotic Process Automation

4.7 Transformation Algorithm

Algorithm 7 Step One (S1)

```
Require: $\Omega$
1: $\mathrm{P} \leftarrow$ new
2: for $\omega \in \Omega$ do
3:     $\rho \leftarrow OTA(\omega)$
4:     $\mathrm{P}.add(\rho)$
5: end for
```

```
6: return P
```

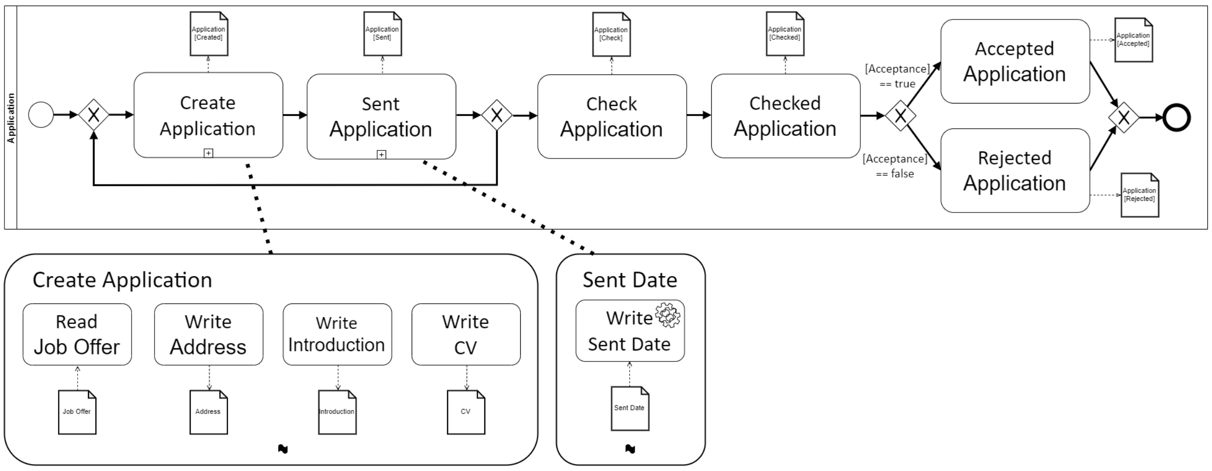

Abbildung 4.4: Initial example

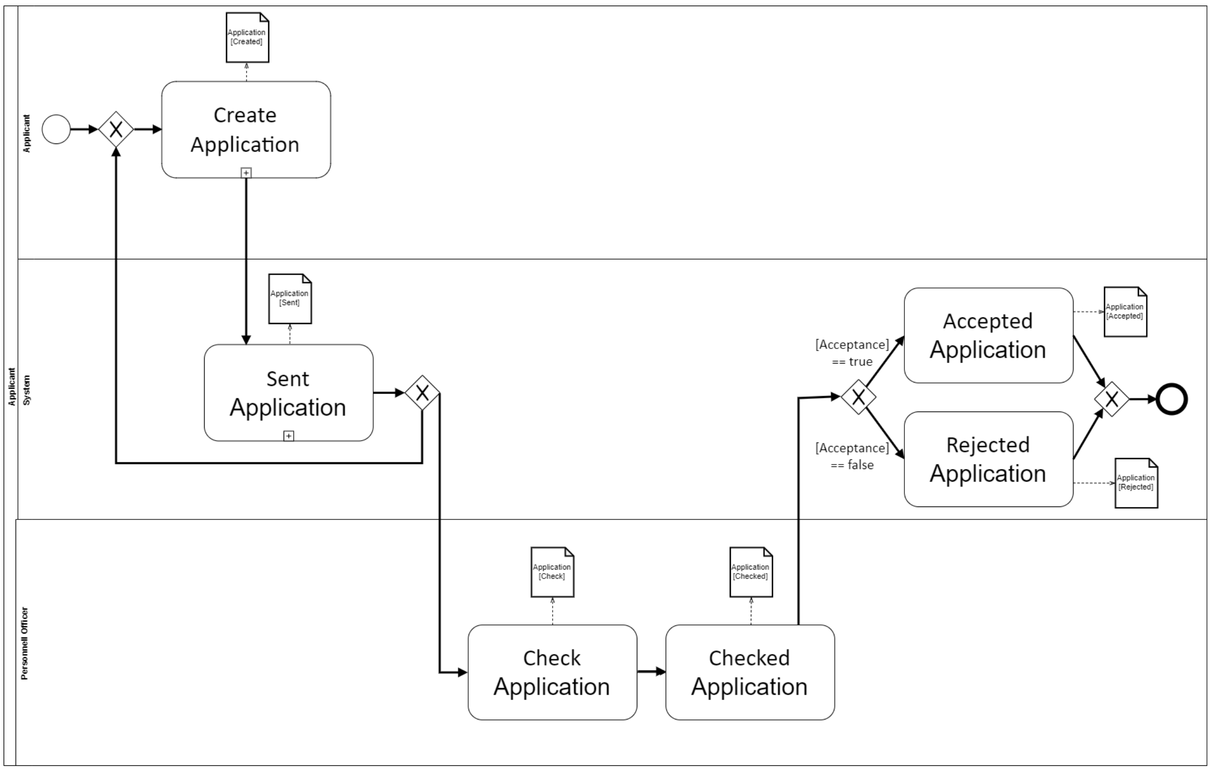

Abbildung 4.5: Initial example

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.

```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