Validating Process Refinement with Ontologies

Yuan Ren[1], Gerd Groener[2], Jens Lemcke[3], Tirdad Rahmani[3], Andreas Friesen[3], Yuting Zhao[1], Jeff Z. Pan[1] and Steffen Staab[2]

[1]University of Aberdeen, [2]University of Koblenz-Landau, [3]SAP AG

Abstract. A crucial task in process management is the validation of process refinements. A process refinement is a process description in a more fine-grained representation. The refinement is either with respect to an abstract model or with respect to component's principle behaviour model. We define process refinement based on the execution set semantics. Predecessor and successor relations of the activities are described in an ontology in which the refinement is represented and validated by concept satisfiability checking.

1 Introduction

The refinement of processes is a common task in process development. A generic process describes the core functionality of an application. A refinement is a transformation of a process into a more specific process description which is developed for a more concrete application and based on more detailed process behaviour knowledge. A key question is whether a process refinement is valid, i.e. the refined process refers to the intended behaviour of the abstract process.

We distinguish between horizontal and vertical refinement. A horizontal refinement is a transformation from an abstract to a more specific model. A vertical refinement is a transformation from a principle behaviour model of a software component to a concrete process model of business. Once a refined process model describes a concrete application process, the relationship to the original generic process is not always obvious due to a number of possibly complex refinement steps. A refinement step includes customisation, extensions and compositions of processes. Therefore it is quite difficult to validate the refinement of a process with respect to an abstract process model.

The validation of a process refinement consists of checking model constraints like execution order constraints. Modelling constraints for the refined process has to cover the constraints of the generic process behaviour and also constraints which emerged during the refinement steps.

In this paper, we use execution set semantics to analyse process refinements. This set-based formalism accounts for a comparison of the process executions of different processes. The comparison of process execution sets is then reduced to the comparison of finite sets of predecessor and successor relations which is realized by subsumption checking. We further use $\mathcal{ALC}$ ontologies to model processes and process constraints, concerning the execution order conditions

and binding of tasks. Finally the refinement checking is reduced to concept satisfiability checking.

In Section 2 we define the problem of process refinement with its graphical syntax, semantics and mathematical foundation. The modelling of processes with the corresponding execution constraints is demonstrated in Section 3, followed by the refinement validation.

2 Process Refinement

2.1 Execution Set Semantics

A process is a non-simple directed graph without multiple edges between two vertices. A vertex is either an activity, a gateway, a start, or an end event. As a graphical process syntax, we use the business process modelling notation (BPMN)[1] due to its wide industry adoption. As an example, in Fig. 1, (a) shows a BPMN diagram of a simple flow which consists of two activities between the Start and End nodes; (b) shows the usage of exclusive gateway (◈) indicating that the flow can go through one and only one of its branches, and parallel gateway (◈) indicating that the flow should go through all of its branches; (c) shows the usage of a loop indicating that the flow can go back to previous gateways or activities. In block-based process modelling, gateways always appear in pairs. Thus in this work we consider only pairwise gateways to comply such convention. Also, we assume all the process models are valid by their own.

As a prerequisite to validating, we define the correctness of process refinement based on the execution set semantics [14]. Briefly, the execution set of a process P, denoted by ES_P, is made up of all the valid paths between the Start and the End in P. $ES_{Abstract}$ in Fig. 1a is $\{[\mathsf{AB}]\}$: first A, then B. Any two branches behind an exclusive gateway (◈) cannot occur together in the same path, thus $ES_{Component2}$ is $\{[\mathsf{E}], [\mathsf{EF}]\}$ (see Fig. 1d). When they are behind a parallel gateway (◈), any ordering between activities of the branches are valid, thus $[a_1a_2b_1b_2]$, $[a_1b_1a_2b_2]$ and $[a_1b_1b_2a_2] \in ES_{Specific}$ (see Fig. 1b). The execution set of a process containing a loop is infinite, e. g., $ES_{Component1} = \{[\mathsf{C}], [\mathsf{CDCD}], \ldots\}$ (see Fig. 1c). Consequently, enumerating the execution set is infeasible.

2.2 Process Refinement

Fig. 1 shows a refinement horizontally from abstract to specific while vertically complying with the components' principle behaviour. In our example scenario, Fig. 1a is drawn by a line of business manager to sketch a new hiring process. Fig. 1b is drawn by a process architect who incrementally implements the sketched process. Fig. 1c and d are the principle behaviour models of different components. The challenge is to verify whether the refinement is consistent with the more abstract horizontal and vertical processes.

[1] `http://www.bpmn.org/`

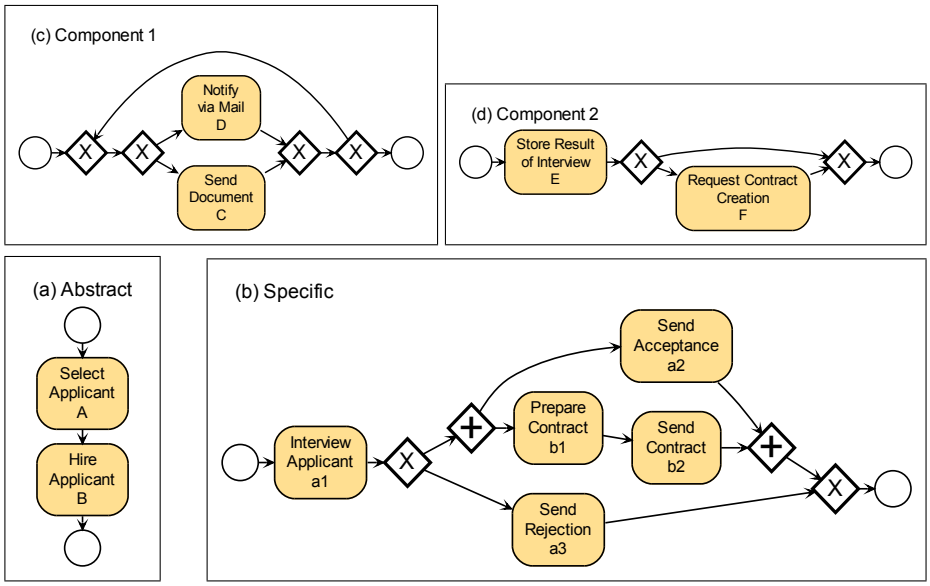

Fig. 1. Wrong process refinement

To facilitate horizontal validation, the process architect has to declare which activities of Fig. 1b implement which activity of Fig. 1a: $\mathrm{hori}(\mathsf{a}_1) = \mathrm{hori}(\mathsf{a}_2) = \mathrm{hori}(\mathsf{a}_3) = \mathsf{A}$, $\mathrm{hori}(\mathsf{b}_1) = \mathsf{B}$, $\mathrm{hori}(\mathsf{b}_2) = \mathsf{B}$. While for vertical validation, the process architect needs to link activities of Fig. 1b to service endpoints given in Fig. 1c and d: $\mathrm{vert}(\mathsf{a}_1) = \mathsf{E}$, $\mathrm{vert}(\mathsf{a}_2) = \mathrm{vert}(\mathsf{a}_3) = \mathsf{D}$, $\mathrm{vert}(\mathsf{b}_1) = \mathsf{F}$, $\mathrm{vert}(\mathsf{b}_2) = \mathsf{C}$. In case such originality relations characterise the refinement, we assume they are always correct in this work.

Correct horizontal refinement. We say that a process Q is a correct *horizontal* refinement of a process P if $ES_Q \subseteq ES_P$ after the following transformations.

1. **Renaming.** Replace all activities in each execution of ES_Q by their originators (function hori()). Renaming the execution set $\{[\mathsf{a}_1\mathsf{a}_2\mathsf{b}_1\mathsf{b}_2], [\mathsf{a}_1\mathsf{b}_1\mathsf{b}_2\mathsf{a}_2], [\mathsf{a}_1\mathsf{b}_1\mathsf{a}_2\mathsf{b}_2], [\mathsf{a}_1\mathsf{a}_3]\}$ of Fig. 1b yields $\{[\mathsf{AABB}], [\mathsf{ABBA}], [\mathsf{ABAB}], [\mathsf{AA}]\}$.
2. **Decomposition.** Replace all pairs of duplicate activities by a single activity in each execution of ES_Q. For Fig. 1b this yields $\{[\mathsf{AB}], [\mathsf{ABA}], [\mathsf{ABAB}], [\mathsf{A}]\}$.

As $\{[\mathsf{AB}]\} \not\supseteq \{[\mathsf{AB}], [\mathsf{ABA}], [\mathsf{ABAB}], [\mathsf{A}]\}$, Fig. 1b is a wrong horizontal refinement of Fig. 1a. The cause is the potentially inverted order of AB by $\mathsf{b}_1\mathsf{a}_2$ or $\mathsf{b}_2\mathsf{a}_2$ in Fig. 1b and that a_3 can be the last activity in Fig. 1b, whereas A must always be followed by B in Fig. 1a.

Correct vertical refinement. We say that a process Q is a correct *vertical* refinement of a process P if $ES_Q \subseteq ES_P$ after the following transformations.

1. **Renaming.** Replace all activities in each execution of ES_Q by their grounds (function vert()). Renaming the execution set $\{[\mathsf{a_1a_2b_1b_2}], [\mathsf{a_1b_1b_2a_2}], [\mathsf{a_1b_1a_2b_2}], [\mathsf{a_1a_3}]\}$ of Fig. 1b yields $\{[\mathsf{EDFC}], [\mathsf{EFCD}], [\mathsf{EFDC}], [\mathsf{ED}]\}$.
2. **Reduction.** Remove all activities in each execution of ES_Q that do not appear in P. For our example, reduction with respect to Component 1 yields $\{[\mathsf{DC}], [\mathsf{CD}], [\mathsf{D}]\}$. Reduction with respect to Component 2 yields $\{[\mathsf{EF}], [\mathsf{E}]\}$.

As $\{[\mathsf{C}], [\mathsf{D}], [\mathsf{CC}], [\mathsf{CD}], [\mathsf{DC}], [\mathsf{DD}], \ldots\} \supseteq \{[\mathsf{DC}], [\mathsf{CD}], [\mathsf{D}]\}$ and $\{[\mathsf{EF}], [\mathsf{E}]\} \supseteq \{[\mathsf{EF}], [\mathsf{E}]\}$, Fig. 1b is a correct vertical refinement of both Component 1 and 2.

As enumerating the execution sets for validation is infeasible, our solution works with descriptions in ontology instead of using the execution sets themselves.

3 Validation with Ontologies

In this section, we present our solution of validating process refinement with ontologies in detail. We accomplish the transformations from the last section by reductions on the process diagram and execution sets, represent the refinement with an ontology, and validate the refinement by concept satisfiability checking.

3.1 Refinement reduction

Because the execution set may contain infinite number of executions, e.g. $ES_{component1}$, it's difficult to directly compare them. Therefore we reduce the subsumption between infinitely large execution sets into subsumption between finite *predecessor* and *successor* sets that are also obtained from the process diagram and show that the transformations on the execution sets are equivalent to transformations on the process diagrams or the predecessor and successor sets.

As we can see from $ES_{Specific}$, the execution ordering relations between branches of a parallel gateway are implicit in the original process. In order to make them explicit, we first reduce a process diagram by the following operation:

A parallel gateway $\diamondsuit([B_1|P_1], \ldots, [B_n|P_n])$ is reduced to an exclusive gateway $\diamondsuit\big(\big[B_1\big|\diamondsuit(P_1, [B_2|P_2], \ldots, [B_n|P_n])\big], \ldots, \big[B_n\big|\diamondsuit([B_1|P_1], \ldots, [B_{n-1}|P_{n-1}], P_n)\big]\big)$, where the notation $[B|P]$ represents a path with B the head and a subpath P the tail.

We repeatedly apply above operation until there's no parallel gateway in the diagram. Such an iteration will always terminate due to the finite size of the process. The resulting BPMN graph is called the *Execution Diagram* of P, denoted by ED_P (see Fig. 2). An activity $a \in P$ may have multiple appearance in ED_P. We use one more subscript j to differentiate them, i.e. $a_2 1$, $a_2 2$ w.r.t. a_2. This transformation may increase the size of the process model exponentially, e.g. a pair of parallel gateways containing n branches of one activity will be transformed into a pair of exclusive gateway containing $n!$ branches of n activity.

Obviously, given a process P, its execution set ES_P is the set of all routes between Start and End in its execution diagram ED_P after replacing duplicated activities with their original names.

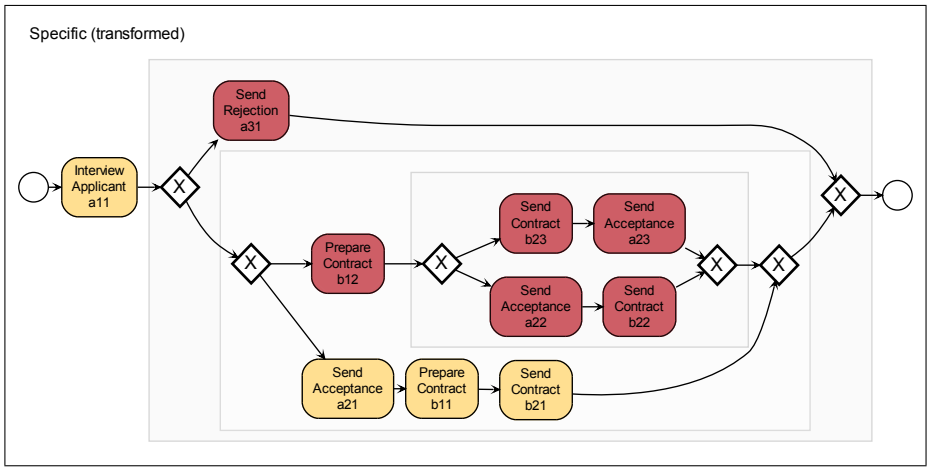

Fig. 2. Execution diagram for Fig. 1b

We further reduce ED_P from a local perspective. For each activity appearance a in ED_P, its *Predecessor Set* $PS_P(a)$ is the set of all the activities going to a through a direct link or a sequence containing only gateways. And its *Successor Set* $SS_P(a)$ is the set of all the activities going from a through a direct link or a sequence containing only gateways.

Obviously, according to the definition, for each x, y such that $x \in PS_P(a_j)$, $y \in SS_P(a_j)$, $[x, a_j, y]$ is a valid subpath of activities in ED_P and thus $[x, a, y]$ is a valid subpath of activities in P. Because the ED_P is finite, $PS_P(a_j)$ and $SS_P(a_j)$ are also finite for any $a_j \in P$. Follow our example, $PS(a_{31}) = \{a_{11}\}$ and $SS(a_{31}) = \{End\}$.

By obtaining these two sets for all the activities, the relation between two execution sets can be characterised by the following theorem:

Theorem 1. *$ES_Q \subseteq ES_P$ iff $\forall a \in Q$, $PS_Q(a) \subseteq PS_P(a)$ and $SS_Q(a) \subseteq SS_P(a)$.*

Proof. (1) For the $\rightarrow$ direction the lhs $ES_Q \subseteq ES_P$ holds. Given $a \in Q$ and $b \in PS_Q(a)$, we demonstrate that $b \in PS_P(a)$ holds: $[b, a]$ is a subpath of some $X \in ES_Q$, from precondition it follows $[b, a]$ is a subpath of $X \in ES_P$ and the definition of PS leads to $b \in PS_P(a)$. Correspondingly for $c \in SS_Q(a)$, it follows from definition $[a, c]$ is a subpath of some $Y \in ES_Q$ and from precondition follows $[a, c]$ is a subpath of $Y \in ES_P$. The definition of SS leads to $c \in SS_P(a)$.
(2) For the $\leftarrow$ direction the rhs holds and we demonstrate that $ES_Q \subseteq ES_P$ follows. Consider an execution $s \in ES_Q$, $s = a_1, \ldots, a_n$. Activity a_{i-1} is predecessor of a_i. For $i = 2, \ldots, n$ the following implication holds: $PS_Q(a_i) \subseteq PS_P(a_i) \Rightarrow [a_{i-1}, a_i]$ is a subpath of some $X \in ES_P$. For $i = 1, \ldots, n-1$ the following implication holds: $SS_Q(a_i) \subseteq SS_P(a_i) \Rightarrow [a_i, a_{i+1}]$ is a subpath of some $Y \in ES_P$, therefore sequentially connect the subpathes $[a_1, \ldots, a_n] \in ES_P$ holds.

Therefore, we reduce the process refinement w.r.t. execution set semantics into the subsumption checking of finite predecessor and successor sets. We then show that the transformation operations of execution sets can be equivalently performed on the original process diagrams and the predecessor and successor sets obtained from them:

- **Reduction** on the process diagrams has the same effect on the execution sets. That means, given a component model P and a process model Q, if we reduce Q into Q' by removing all the activities that do not appear in P, and link their predecessors and successors directly, the resulting $ES_{Q'}$ will be the same as the reduction of ES_Q with respect to P.
- **Renaming** can also be directly performed on the process diagram, i.e. $ES_P[a \rightarrow A] = ES_{P[a \rightarrow A]}$. Thus, the renaming can be performed on the predecessor and successor sets as well, i.e. $PS_P(x)[a \rightarrow A] = PS_{P[a \rightarrow A]}(x)$ $(SS_P(x)[a \rightarrow A] = SS_{P[a \rightarrow A]}(x))$.
- **Decomposition** can be done on the predecessor and successor sets as well. Particularly, $ES_{Q'} \subseteq ES_P$, where $ES_{Q'}$ is ES_Q after decomposition of all the sequential x grouped into a single x, *iff* $\forall a \neq x$, $PS_Q(a) \subseteq PS_P(a)$, $SS_Q(a) \subseteq SS_P(a)$ and $\forall x$, $PS_Q(x) \cup \{x\} \subseteq PS_P(x)$ and $SS_Q(x) \cup \{x\} \subseteq SS_P(x)$. This interprets the decomposition as, any x that should be grouped, can go not only from predecessors of x, but also another appearance of x, and can go not only to successors of x, but also another appearance of x.

Thus, above analysis implies that:

- For horizontal refinement, we can first obtain the predecessor and successor sets of activities, and then perform the **Renaming** and **Decomposition** on these sets, and then check the validity.
- For vertical refinement, we can first perform the **Reduction** on processes, then obtain the predecessor and successor sets and perform the **Renaming** on these sets, and finally check the validity.

In this paper, we perform **Reduction** directly on the process diagram and obtain the predecessor and successor sets from its execution diagram, then perform **Renaming** and **Decomposition** with ontologies and check the validity with reasoning.

3.2 Refinement representation

In this section we represent the predecessor and successor sets of activities with ontologies. In such ontologies, activities are represented by concepts. The predecessors and successors relations are described by two roles *from* and *to*, respectively. Composition of activities in horizontal refinement is described by role *compose*. Grounding of activities in vertical refinement is described by role *groundedTo*. The last two don't have essential logic difference in the problem of process refinement but have different semantics in the process model, so that we distinguish between them. Then, for a set S containing predecessors or successors, we define four operators for translations as follows:

Definition 1. *:*

Pre-refinement-from operator $\boldsymbol{Pr}_{from}(S) = \forall from. \bigsqcup_{x \in S} x$
Pre-refinement-to operator $\boldsymbol{Pr}_{to}(S) = \forall to. \bigsqcup_{y \in S} y$
Post-refinement-from operator $\boldsymbol{Ps}_{from}(S) = \bigsqcap_{x \in S} \exists from.x$
Post-refinement-to operator $\boldsymbol{Ps}_{to}(S) = \bigsqcap_{y \in S} \exists to.y$

The effect of the above operators in refinement checking can be characterised by the following theorem:

Theorem 2. $PS_Q(a) \subseteq PS_P(a)$ iff
$Disjoint(x|x \in P \cup Q)$ *infers that* $\boldsymbol{Pr}_{from}(PS_P(a)) \sqcap \boldsymbol{Ps}_{from}(PS_Q(a))$ *is satisfiable.*

$SS_Q(a) \subseteq SS_P(a)$ iff
$Disjoint(x|x \in P \cup Q)$ *infers that* $\boldsymbol{Pr}_{to}(SS_P(a)) \sqcap \boldsymbol{Ps}_{to}(SS_Q(a))$ *is satisfiable.*

For sake of a shorter presentation, we only prove the first part of the theorem. The proof for the second part is appropriate to the first part.

Proof. (1) We demonstrate $\rightarrow$ direction with a proof by contraposition. The disjointness of activities holds. Supposed the rhs is unsatisfiable, i.e. $\mathbf{Pr}_{from}(PS_P(a)) \sqcap \mathbf{Ps}_{from}(PS_Q(a))$ is unsatisfiable. Obviously, both concept definitions on its own are satisfiable, since $\mathbf{Pr}_{from}(PS_P(a))$ is just a definition with one all-quantified role followed by a union of (disjoint) concepts. The concept definition behind this expression is $\forall from. \bigsqcup_{x \in PS_P(a)} x$ which restricts the range of $from$ to all concepts (activities) of $PS_P(a)$. $\mathbf{Ps}_{from}(PS_Q(a))$ is a concept intersection which only consists of existential quantifiers and the same $from$ role. This definition is also satisfiable. Therefore the unsatisfiability is caused by the intersection of both definitions. In $\mathbf{Ps}_{from}(PS_Q(a))$ the same role $from$ is used and the range is restricted by $\mathbf{Pr}_{from}(PS_P(a))$. Therefore the contradiction is caused by one activity $b \in PS_Q(a)$ which is not in $PS_P(a)$, but this is a contradiction to the precondition $PS_Q(a) \subseteq PS_P(a)$.
(2) For the $\leftarrow$ direction we assume that the rhs
$Disjoint(x|x \in P \cup Q)$ infers that $\mathbf{Pr}_{from}(PS_P(a)) \sqcap \mathbf{Ps}_{from}(PS_Q(a))$ is satisfiable. For each activity $b \in PS_Q(a)$ we show that b is also in $PS_P(a)$: $b \in PS_Q(a)$ and $\mathbf{Ps}_{from}(PS_Q(a))$ is satisfiable, the intersection contains the term $\exists from.b$. Since $\mathbf{Pr}_{from}(PS_P(a))$ is satisfiable and is also the range restriction of $from$, it follows that $b \in PS_P(a)$.

This theorem indicates that the refinement checking of processes can be reduced to the satisfiability checking of concepts in an ontology. In the following, we present the representation of horizontal refinement and vertical refinement.

Horizontal Refinement For conciseness of presentation, we always have a pre-refinement process and a post-refinement process and we refine one activity in each step (this activity may have multiple appearance after conversion to the execution diagram). We assume P the pre-refinement process, Q the post-refinement process and z the activity being refined. For each z_j we define

$component\_z_j \equiv \exists compose.z_j$. The simultaneous refinement of multiple activities can be done in a similar manner of single refinement. Then we construct an ontology $\mathcal{O}_{P\to Q}$ with following axioms:

1. for each activity $a \in Q$ and $hori(a) = z_j$, $a \sqsubseteq \exists compose.z_j$
   These axioms represent the composition of activities with concept subsumptions, which realise **Renaming** in horizontal refinement. For example, $b_{11} \sqsubseteq \exists compose.B$ and $a_{31} \sqsubseteq \exists compose.A$.
2. for each $a \in Q$ and a is not refined from any z_j
   $a \sqsubseteq \mathbf{Pr}_{from}(PS_P(a))[z_j \to component\_z_j]$,
   $a \sqsubseteq \mathbf{Pr}_{to}(SS_P(a))[z_j \to componennt\_z_j]$,
   These axioms represent the predecessor and successor sets of all the unrefined activities in the pre-refinement process. Because in the post-refinement process, any activity refined from z_j will be considered as a subconcept of $component\_z_j$, we replace the appearance of each z_j by corresponding $component\_z_j$. For example, $Start \sqsubseteq \forall to.component\_A$.
3. for each $z_j \in P$,
   $component\_z_j \sqsubseteq \mathbf{Pr}_{from}(PS_P(z_j) \cup \{component\_z_j\})[z_j \to componennt\_z_j]$,
   $component\_z_j \sqsubseteq \mathbf{Pr}_{to}(SS_P(z_j) \cup \{component\_z_j\})[z_j \to componennt\_z_j]$,
   These axioms represent the predecessor and successor sets of all the refined activities in the pre-refinement process. Due to the mechanism of **Decomposing**, we add the corresponding $component\_z_j$ to their predecessor and successor sets, and replace the z_j with $component\_z_j$ for the same reason as before. For example, $component\_A \sqsubseteq \forall from.(Start \sqcup component\_A)$, $component\_A \sqsubseteq \forall to.(component\_B \sqcup component\_A)$, $component\_B \sqsubseteq \forall from.(component\_A \sqcup component\_B)$ and $component\_B \sqsubseteq \forall to.(End \sqcap component\_B)$.
4. for each $a \in Q$, $a \sqsubseteq \mathbf{Ps}_{from}(PS_Q(a))$ and $a \sqsubseteq \mathbf{Ps}_{to}(SS_Q(a))$,
   These axioms represent the predecessor and successor sets of all the activities in the post-refinement process. For example, $a_{31} \sqsubseteq \exists to.End$, $b_{11} \sqsubseteq \exists to.a_{21} \sqcap \exists to.b_{22}$
5. for each $z_j \in P$, $Disjoint(a|a \in Q$ and $Hori(a) = z_j)$
   These axioms represent the uniqueness of all the sibling activities refined from the same z_j. For example, $Disjoint(a_{11}, a_{21}, a_{22}, a_{23}, a_{31})$.
6. $Disjoint($ all the activity in P, and all the $component\_z_j)$.
   This axiom represents the uniqueness of all the activities before refinement. For example, $Disjoint(Start, End, A, B, component\_A, component\_B)$.

With above axioms, ontology $\mathcal{O}_{P\to Q}$ is a representation of the horizontal refinement from P to Q by describing the predecessor and successor sets of corresponding activities with axioms.

Vertical Refinement Similar as horizontal refinement, assume we have principle behaviour model P and a concrete process model Q, which is reduced w.r.t P to eliminate ungrounded activities. Any activity in Q can be grounded to some activity in P. Thus, after reduction, $\forall a \in P, \exists b \in Q$ that b is grounded

to a, and vice versa. Therefore for each $x_j \in P$, we define $grounded\_x_j \equiv \exists groundedTo.x_j$.Then we construct an ontology $\mathcal{O}_{P\to Q}$ with following axioms:

1. for each activity $a \in Q$ and $vert(a) = x_j$, $a \sqsubseteq \exists groundedTo.x_j$
   These axioms represent the grounding of activities by concept subsumptions, which realise the **Renaming** in vertical refinement. For example, $a_{11} \sqsubseteq \exists groundedTo.E$, $b_{11} \sqsubseteq \exists groundedTo.F$.
2. for each $a \in P$
   $grounded\_a \sqsubseteq \mathbf{Pr}_{from}(PS_P(a))[x_j \to grounded\_x_j]$,
   $grounded\_a \sqsubseteq \mathbf{Pr}_{to}(SS_P(a))[x_j \to grounded\_x_j]$,
   These axioms represent the predecessor and successor sets of all the activities in the pre-refinement process. Due the mechanism of **Renaming** we replace all the $x_j \in P$ by $grounded\_x_j$. Because **Decomposition** is not needed in vertical refinement, we stick to the original predecessor and successor sets. These axioms become the constraints on the activities in Q. For example, $grounded\_E \sqsubseteq \forall from.Start$, $grounded\_E \sqsubseteq \forall to.(grounded\_F \sqcup End)$.
3. for each $a \in Q$, $a \sqsubseteq \mathbf{Ps}_{from}(PS_Q(a))$ and $a \sqsubseteq \mathbf{Ps}_{to}(SS_Q(a))$,
   These axioms represent the predecessor and successor sets of all the activities in the post-refinement process. For example, $a_{11} \sqsubseteq \exists from.Start$, $a_{11} \sqsubseteq \exists to.b_{11} \sqcap \exists to.End$. Notice that the ungrounded activities such as a_{31} have been removed from the process so that End becomes a directed successor of a_{11} after reduction.
4. for each $x_j \in P$, $Disjoint(a|a \in Q$ and $vert(a) = x_j)$
   These axioms represent the uniqueness of all the sibling activities refined from the same x_j.
5. $Disjoint$(all the $grounded\_x_j$).
   This axiom represents the uniqueness of all the activities before refinement. For example, $Disjoint(Start, End, grounded\_E, grounded\_F)$.

Both the horizontal and vertical refinement can be represented in DL in a linear complexity. The language is $\mathcal{ALC}$. It's worth to mention that $from$ and to do not need to be inverse roles because the unsatisfiability of activities is caused by the contradiction of universal and existential restrictions.

With above axioms, ontology $\mathcal{O}_{P\to Q}$ is a representation of the refinement from P to Q by describing the predecessor and successor sets of corresponding activities with axioms.

3.3 Concept satisfiability checking

Follow from the theorems and definitions presented before, the relation between the ontology $\mathcal{O}_{P\to Q}$ and the validity of the refinement from P to Q is characterised by the following theorem:

Theorem 3. *The route contains activity a in Q is invalid in the refinement from P to Q,* iff $\mathcal{O}_{P\to Q} \models a \sqsubseteq \bot$.

Proof. For each a in Q the ontology $\mathcal{O}_{P\rightarrow Q}$ contains the axioms $a \sqsubseteq \mathbf{Ps}_{from}(PS_Q(a))$ and $a \sqsubseteq \mathbf{Ps}_{to}(SS_Q(a))$. The axioms $a \sqsubseteq \mathbf{Pr}_{from}(PS_Q(a))$ and $a \sqsubseteq \mathbf{Pr}_{to}(SS_Q(a))$ are derived from the axioms (item 1,2). Depending on the refinement either the axioms $a \sqsubseteq \exists groundedTo.x_j$ and $grounded\_x_j \equiv \exists groundedTo.x_j$ or $a \sqsubseteq \exists compose.z_j$ and $component\_z_j \equiv \exists compose.z_j$ are in the ontology. (1) For the $\rightarrow$ direction the lhs holds, we demonstrate that a is unsatisfiable. Since a is invalid either $PS_Q(a) \not\subseteq PS_P(a)$ or $SS_Q(a) \not\subseteq SS_P(a)$. From Theorem 3 it follows that either $\mathbf{Pr}_{from}(PS_P(a)) \sqcap \mathbf{Ps}_{from}(PS_Q(a))$ or $\mathbf{Pr}_{to}(SS_P(a)) \sqcap \mathbf{Ps}_{to}(SS_Q(a))$ is unsatisfiable and therefore a is unsatisfiable since a is subsumed. (2) The $\leftarrow$ direction is proved by contraposition. Given a is unsatisfiable in $\mathcal{O}_{P\rightarrow Q}$. Assumed a is valid in the refinement then $PS_Q(a) \subseteq PS_P(a)$ and $SS_Q(a) \subseteq SS_P(a)$ holds. From Theorem 3 the satisfiability of $\mathbf{Pr}_{to}(SS_P(a))$, $\mathbf{Pr}_{from}(PS_P(a))$, $\mathbf{Ps}_{from}(PS_Q(a))$ and $\mathbf{Ps}_{to}(SS_Q(a))$ follows which leads to a contradiction to the satisfiability of a.

This theorem has two implications:

1. The validity of a refinement can be checked by the satisfiability of all the name concepts in an ontology;
2. The activities represented by unsatisfiable concepts in the ontology are the source of the invalid refinement.

The complexity of such concept satisfiability checking is, according to the expressive power, ExpTime. we check the satisfiability of the concepts to validate the process refinement. Every unsatisfiable concept is either an invalid refinement or relating to an invalid refinement.

For example, in the horizontal refinement ontology, $a_{31} \sqsubseteq \exists compose.A$, $component\_A \equiv \exists compose.A$ and $component\_A \sqsubseteq \forall to.(component\_B \sqcup component\_A)$ implies that $a_{31} \sqsubseteq \forall to.(component\_B \sqcup component\_A)$. However $a_{31} \sqsubseteq \exists to.End$, and $Disjoint(End, component\_A, component\_B)$ indicates that a_{31} is unsatisfiable. Therefore the top route $[a11, a31]$ in the specific process model is invalid, so as the entire refinement.

While in the vertical refinement ontology, it's easy to infer that all the concepts are satisfiable. Therefore, the process model does comply to the principle behaviour models.

Helped by our analysis, the line of business manager and process architect remodel their processes (Fig. 3). Now, the execution set of Fig. 3a is $\{[\mathsf{AB}], [\mathsf{A}]\}$. Renaming of Fig. 3b's execution set $\{[\mathsf{a_1a_2b_1b_2}], [\mathsf{a_1a_3}]\}$ yields $\{[\mathsf{AABB}], [\mathsf{AA}]\}$. After decomposition, we conclude that Fig. 3b correctly refines the process in Fig. 3a because $\{[\mathsf{AB}], [\mathsf{A}]\} \supseteq \{[\mathsf{AB}], [\mathsf{A}]\}$. As for comparing with the component models, renaming yields $\{[\mathsf{EDFC}], [\mathsf{ED}]\}$. After reduction with respect to $\mathsf{C_1}$ and $\mathsf{C_2}$, we conclude that Fig. 3b correctly grounds on $\mathsf{C_1}$ and $\mathsf{C_2}$ because $\{[\mathsf{C}], [\mathsf{D}], [\mathsf{CC}], [\mathsf{CD}], [\mathsf{DC}], [\mathsf{DD}], \ldots\} \supseteq \{[\mathsf{DC}], [\mathsf{D}]\}$ and $\{[\mathsf{EF}], [\mathsf{E}]\} \supseteq \{[\mathsf{EF}], [\mathsf{E}]\}$.

4 Related Works and Conclusion

Annotations to process models for service behaviour and interaction is described in [4, 12, 11]. However, process refinement and model validation is not considered.

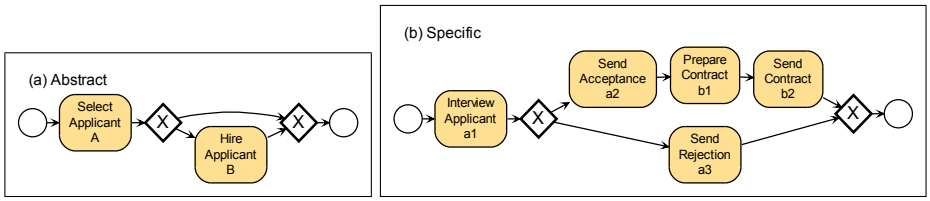

Fig. 3. Correct process refinement

Other models use mathematical formalisms to describe and compare concurrent system behaviour. In [3] a process algebra is described in order to analyse equivalence. The analysed equivalence (trace equivalence) does not distinguish between deterministic and non-deterministic choices. Semantics for BPMN models in process algebra and process refinements are outlined in [13]. The refinement is validated with model checker.

Calculus of communication systems and transition systems is used in other works like in [7, 8]. The π-calculus is a first order calculus for concurrent systems. The bisimulation (equivalence relationship between transition systems) for the π-calculus is described in [9] and in [10] higher-order process calculus is used to analyse bisimulation. Some of these approaches also apply the execution set semantics from [14], but without refinement validation.

In [5] actions and services are described in DL. Actions contain pre- and post-conditions. As inference problems, the realizability of a service, subsumption relation between services and service effect checking is analysed. Services are also described with DL in [2]. The reasoning tasks are checking of pre- and post-conditions. The focus of this work is the reasoning complexity.

The DL $\mathcal{DLR}$ is extended with temporal operators in [1]. Based on this extension, query containment for specified (temporal) properties is analysed. In [6] the DL $\mathcal{ALC}$ is extended with the temporal logics LTL and CTL. Still, neither of them considers process modelling and refinements.

In this paper, we have devised a method to checking process refinement w.r.t. execution set semantics. We restricted our solution to a commonly used subset of BPMN [15], which we may extend in the future. We perform a topological transformation of process models and translate them into $\mathcal{ALC}$ ontologies. The refinement checking can be reduced to concept unsatisfiability checking. In the future, we will implement and evaluate our approach, and extend it to deal with more constraints.

Acknowledgements

This work has been supported by the European Project Marrying Ontologies and Software Technologies (MOST ICT 2008-216691).

References

1. A. Artale, E. Franconi, F. Wolter, and M. Zakharyaschev. A Temporal Description Logic for Reasoning over Conceptual Schemas and Queries. *Lecture notes in computer science*, pages 98–110, 2002.
2. F. Baader, C. Lutz, M. Milicic, U. Sattler, and F. Wolter. A Description Logic Based Approach to Reasoning about Web Services. In *Proceedings of the WWW 2005 Workshop on Web Service Semantics (WSS2005)*, Chiba City, Japan, 2005.
3. A.J. Cowie. *The Modelling of Temporal Properties in a Process Algebra Framework*. PhD thesis, University of South Australia, 1999.
4. Markus Fronk and Jens Lemcke. Expressing semantic Web service behavior with description logics. In *Semantics for Business Process Management Workshop at ESWC*, 2006.
5. C. Lutz and U. Sattler. A Proposal for Describing Services with DLs. In *Proceedings of the 2002 International Workshop on Description Logics*, 2002.
6. C. Lutz, F. Wolter, and M. Zakharyaschev. Temporal description logics: A survey. In *Temporal Representation and Reasoning, 2008. TIME'08. 15th International Symposium on*, pages 3–14, 2008.
7. R. Milner. *A Calculus of Communicating Systems*. Springer LNCS, 1980.
8. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
9. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes. *Information and Computation*, pages 41–77, 1992.
10. Davide Sangiorgi. Bisimulation for Higher-Order Process Calculi. *Information and Computation*, 131:141–178, 1996.
11. I. Weber, J. Hoffmann, and J. Mendling. Semantic Business Process Validation. In *Proc. of International workshop on Semantic Business Process Management*, 2008.
12. I. Weber, Joerg Hoffmann, and Jan Mendling. Beyond Soundness: On the Correctness of Executable Process Models. In *Proc. of European Conference on Web Services (ECOWS)*, 2008.
13. P.Y.H. Wong and J. Gibbons. A process-algebraic approach to workflow specification and refinement. *Lecture Notes in Computer Science*, 4829:51, 2007.
14. George M. Wyner and Jintae Lee. Defining specialization for process models. In *Organizing Business Knowledge: The MIT Process Handbook*, chapter 5, pages 131–174. MIT Press, 2003.
15. Michael zur Muehlen and Jan Recker. How much language is enough? Theoretical and practical use of the Business Process Modeling Notation. In Zohra Bellahsene and Michel Léonard, editors, *CAiSE*, volume 5074 of *Lecture Notes in Computer Science*, pages 465–479. Springer, 2008.