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 with respect to either an abstract model or a 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 can be validated by concept satisfiability checking.

1 Introduction

With the growing interest about applying semantic web technologies on business process modelling, many frameworks and ontological models have been proposed to facilitate a more unified semantic representation [5, 6].

In model-driven software development, process models are usually created and refined on different levels of abstraction. 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. In this procedure, the refined process should refer to the intended behaviour of the abstract process and satisfies behaviour constraints. To check and ensure the consistency of refinement becomes a crucial issue in process management. Currently, such consistency check is mainly done manually and few methods have been investigated to help automation. Hence the validation is error-prone, time-consuming and increases the costs during the development cycle.

In this paper, we use execution set semantics to describe two types of process refinements and present an ontological approach to represent and check them. We first apply topological transformations to reduce the refitment checking w.r.t. execution set semantics into checking of predecessors and successors of process elements. Then we encode process models into OWL DL ontologies. Finally we show that the refinement checking on the process models can be accomplished by concept ussatisfiability checking in the ontology. We implemented our approach and conducted performance evaluation on a set of randomly generated process models. Experiment results showed that, 80% of the refinement validation tasks

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

can be performed within 1s, which is significantly faster than manually consistency checking and the correctness of the validation is guaranteed.

The rest of the paper is organised as follows: in Sec.2 we define the problem of process refinement with its graphical syntax, semantics and mathematical foundation. The representation and validation of processes with the corresponding execution constraints is demonstrated in Sec.3. In Sec.4 we present the evaluations and in Sec.5 we review related works and conclude the paper.

2 Preliminary

In this section, we introduce preliminary knowledge about process models, process refinement w.r.t. execution set semantics and DL-based ontologies.

Syntax of Process Models A process model—or short: process—is a non-simple directed graph $\mathrm{P} = \langle \mathrm{E}, \mathrm{V} \rangle$ without multiple edges between two vertices. As a graphical representation, we use the business process modelling notation (BPMN: `http://www.bpmn.org/`) due to its wide industry adoption. However, we consider a normal form of process models for the sake of this paper as opposed to the full set of partly redundant constructs in BPMN.

In our definition, vertices (V) include activities, gateways ($\mathrm{A}, \mathrm{G} \subseteq \mathrm{V}$), and the specific vertices start and end event ($v_0, v_{\mathrm{end}} \in \mathrm{V}$). Fig. 1a shows a BPMN diagram which consists of two activities between the start and end events.

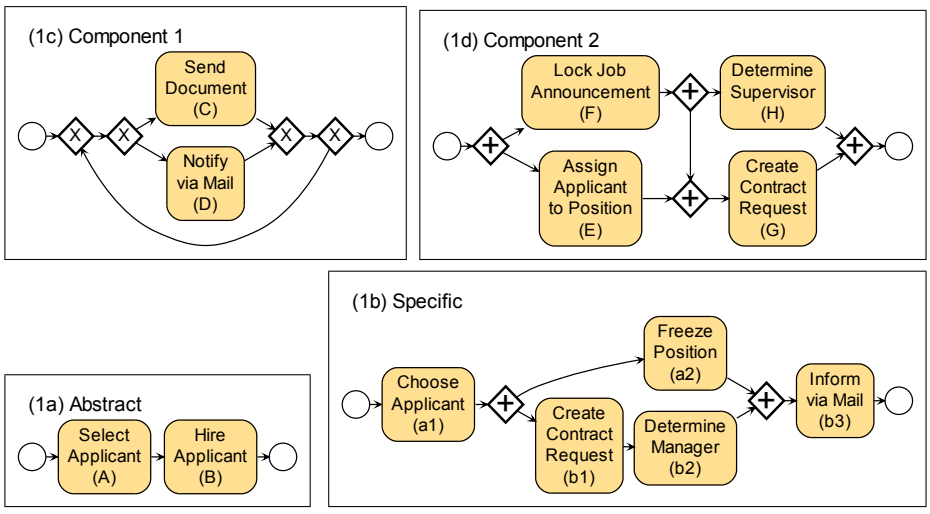

Fig. 1. Wrong process refinement

A gateway is either opening or closing ($\mathrm{G}^{\mathrm{O}}, \mathrm{G}^{\mathrm{C}} \subseteq \mathrm{G}$), and either exclusive or parallel ($\mathrm{G}^{\text{✕}}, \mathrm{G}^{\text{✚}} \subseteq \mathrm{G}$). The process models (c) and (d) in Fig. 1 contain

exclusive and parallel gateways, respectively. We call a process *normal* if it does not contain parallel gateways ($G^{\diamondsuit} = \emptyset$)—as, for example, process model (c).

The edge set (E) is a binary relation on V. We define the predecessor and the successor functions of each $v_1 \in V$ as follows: $pre(v_1) := \{v_2 \in V \mid (v_2, v_1) \in E\}$, $suc(v_1) := \{v_3 \in V \mid (v_1, v_3) \in E\}$. The start (end) event has no predecessor (successor): $|pre(v_0)| = |suc(v_{end})| = 0$ and exactly one successor (predecessor): $|suc(v_0)| = |pre(v_{end})| = 1$. Each open gateway $o \in G^{O}$ (close gateway $c \in G^{C}$) has exactly one predecessor (successor): $|pre(o)| = |suc(c)| = 1$. Each activity $a \in A$ has exactly one predecessor and successor: $|pre(a)| = |suc(a)| = 1$. We can then construct gateway-free predecessor and successor sets as follows:

$$PS(v_1) := \{v_2 \in A \mid v_2 \in pre(v_1) \textit{ or } \exists u \in G \textit{ s.t. } u \in pre(v_1) \textit{ and } v_2 \in PS(u)\}$$
$$SS(v_1) := \{v_3 \in A \mid v_3 \in suc(v_1) \textit{ or } \exists u \in G \textit{ s.t. } u \in suc(v_1) \textit{ and } v_3 \in SS(u)\}$$

These two definitions make gateways "transparent" to ordering relations. For example in Fig.1b, $SS(a1) = \{b1, a2\}$, in Fig.1c, $PS(C) = \{C, D\}$.

Execution Set Semantics of Process Models We define the semantics of a process model using the execution set semantics [18]. An execution is a *proper* sequence of activities ($a_i \in A$): $[a_1 a_2 \dots a_n]$. A proper sequence is obtained by simulating token flow through a process model. A token is associated to exactly one vertex or edge. Initially, there is exactly one token, associated to the start event. Tokens can be created and consumed following the rules below. Whenever a token is created in an activity, the activity is appended to the sequence. Exactly one of the following actions is performed at a time:

- For creating a token in an activity or in the end event $v_1 \in A \cup \{v_{end}\}$, exactly one token must be consumed from the incoming edge $(v_2, v_1) \in E$.
- Exactly one token must be removed from an activity or from the start event $v_1 \in A \cup \{v_0\}$ in order to create one token in the leaving edge $(v_1, v_2) \in E$.
- For creating a token in a parallel close gateway $g \in (G^{\diamondsuit} \cap G^{C})$, exactly one token must be consumed from every incoming edge $(v, g) \in E$.
- For creating a token in an exclusive close gateway $g \in (G^{\diamondsuit} \cap G^{C})$, exactly one token must be consumed from exactly one incoming edge $(v, g) \in E$.
- Exactly one token must be removed from a close gateway $g \in G^{C}$ in order to create one token in the leaving edge $(g, v) \in E$.
- For creating a token in an open gateway $g \in G^{O}$, exactly one token must be consumed from the incoming edge $(v, g) \in E$.
- Exactly one token must be removed from a parallel open gateway $g \in (G^{\diamondsuit} \cap G^{O})$ in order to create one token in each leaving edge $(g, v) \in E$.
- Exactly one token must be removed from an exclusive open gateway $g \in (G^{\diamondsuit} \cap G^{O})$ in order to create one token in exactly one leaving edge $(g, v) \in E$.

If none of the above actions can be performed, simulation has ended. The result is a proper sequence of activities—an execution. It is to be noted that each

execution is finite. However, there may be an infinite number of executions for a process model. The execution set of a process model P, denoted by ES_P, is the (possibly infinite) set of all proper sequences of the process model.

For example, ES_{1a} for process (a) in Fig. 1 is $\{[\mathsf{AB}]\}$: first A, then B (for brevity, we refer to an activity by its short name, which appears in the diagrams in parenthesis). Process (b) contains parallel gateways (⊕) to express that some activities can be performed in any order: $ES_{1b} = \{[\mathsf{a_1a_2b_1b_2b_3}], [\mathsf{a_1b_1a_2b_2b_3}], [\mathsf{a_1b_1b_2a_2b_3}]\}$. Exclusive gateways (⊗) are used in process (c) both to choose from the two activities and to form a loop: $ES_{1c} = \{[\mathsf{C}], [\mathsf{D}], [\mathsf{CC}], [\mathsf{CD}], [\mathsf{DC}], [\mathsf{DD}], \ldots\}$. Process (d) shows that gateways can also occur in a non-block-wise manner: $ES_{1d} = \{[\mathsf{EFGH}], [\mathsf{EFHG}], [\mathsf{FHEG}], [\mathsf{FEGH}], [\mathsf{FEHG}]\}$.

Correct Process Refinement For refinement validation we have to distinguish between horizontal and vertical refinement. A horizontal refinement is a transformation from an abstract to a more specific model which contains the decomposition of activities. A vertical refinement is a transformation from a principle behaviour model of a component to a concrete process model for an application. The validation have to account for both refinements.

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.

To facilitate horizontal validation, the process architect has to declare which activities of Fig. 1b implement which activity of Fig. 1a: $\mathrm{hori}(\mathsf{a_1}) = \mathrm{hori}(\mathsf{a_2}) = \mathsf{A}$, $\mathrm{hori}(\mathsf{b_1}) = \mathrm{hori}(\mathsf{b_2}) = \mathrm{hori}(\mathsf{b_3}) = \mathsf{B}$. For vertical validation, the process architect needs to link activities of Fig. 1b to service endpoints given in Fig. 1c and d: $\mathrm{vert}(\mathsf{a_1}) = \mathsf{E}$, $\mathrm{vert}(\mathsf{a_2}) = \mathsf{F}$, $\mathrm{vert}(\mathsf{b_1}) = \mathsf{G}$, $\mathrm{vert}(\mathsf{b_2}) = \mathsf{H}$, $\mathrm{vert}(\mathsf{b_3}) = \mathsf{D}$.

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 $\{[\mathsf{a_1a_2b_1b_2b_3}], [\mathsf{a_1b_1a_2b_2b_3}], [\mathsf{a_1b_1b_2a_2b_3}]\}$ of Fig. 1b yields $\{[\mathsf{AABBB}], [\mathsf{ABABB}], [\mathsf{ABBAB}]\}$.
2. **Decomposition.** Replace all sequences of equal activities by a single activity in each execution of ES_Q. For Fig. 1b this yields $\{[\mathsf{AB}], [\mathsf{ABAB}]\}$.

As $\{[\mathsf{AB}]\} \not\supseteq \{[\mathsf{AB}], [\mathsf{ABAB}]\}$, Fig. 1b is a wrong horizontal refinement of Fig. 1a. The cause is the potentially inverted order of AB by $\mathsf{b_1a_2}$ or $\mathsf{b_2a_2}$ in Fig. 1b.

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 $\{[\mathsf{a_1a_2b_1b_2b_3}], [\mathsf{a_1b_1a_2b_2b_3}], [\mathsf{a_1b_1b_2a_2b_3}]\}$ of Fig. 1b yields $\{[\mathsf{EFGHD}], [\mathsf{EGFHD}], [\mathsf{EGHFD}]\}$.

2. **Reduction.** Remove all activities in each execution of ES_Q that do not appear in P. For our example, reduction with respect to Fig. 1c yields $\{[\mathsf{D}]\}$. Reduction with respect to Fig. 1d yields $\{[\mathsf{EFGH}], [\mathsf{EGFH}], [\mathsf{EGHF}]\}$.

Fig. 1b is a correct vertical refinement of Fig. 1c because $\{[\mathsf{C}], [\mathsf{D}], [\mathsf{CC}], [\mathsf{CD}], [\mathsf{DC}], [\mathsf{DD}], \ldots\} \supseteq \{[\mathsf{D}]\}$ and a wrong vertical refinement of Fig. 1d because $\{[\mathsf{EFGH}], [\mathsf{EFHG}], [\mathsf{FHEG}], [\mathsf{FEGH}], [\mathsf{FEHG}]\} \not\supseteq \{[\mathsf{EFGH}], [\mathsf{EGFH}], [\mathsf{EGHF}]\}$. The cause for the wrong refinement is the potentially inverted execution of FG by $\mathsf{b_1a_2}$ in Fig. 1b.

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

Description Logics and Ontologies DL-based ontologies have been widely applied as knowledge formalism for the semantic web. An ontology usually consists of a terminology box (TBox) and an assertion box (ABox). In TBox the domain is described by concepts and roles with DL constructs. In this paper, we use DL $\mathcal{ALC}$. Its concepts are inductively defined by following constructs:

$$\top, \bot, A, \neg C, C \sqcap D, C \sqcup D, \exists r.C, \forall r.D$$

where $\top$ is the super concept of all concepts; $\bot$ means nothing; A is an atomic concept; C and D are general concepts and r is an atomic role. In DL, the subsumption between two concepts C and D is depicted as $C \sqsubseteq D$. If two concepts mutually subsume each other, they are equivalent, depicted by $C \equiv D$. When a concept can not be instantiated in any model, i.e., $C \sqsubseteq \bot$, it is unsatisfiable. Two concepts are disjoint if $C \sqsubseteq \neg D$. In this paper we write $Disjoint(C_1, C_2, \ldots, C_n)$ to denote that all these concepts disjoint with one another.

3 Validation with Ontologies

In this section, we present our solution of validating process refinement in detail. We first eliminate all the parallel gateways in a process, then translate such a process into ontologies based on the predecessor and successor sets of activities, finally we show that the refinement checking can be reduced to concept unsatisfiability checking

3.1 Process Transformation

As we can see from ES_{1c}, the execution ordering relations between successors of some $g \in G^O$ are implicit in the original process. For example, $b1$ and $a2$ does not have any explicit edge, the semantics of parallel gateway still implies that $b1a2$ or $a2b1$ must appear in some execution. In order to make such relations explicit, we eliminate all the parallel gateways while retain the execution set. Our strategy is to generate exclusive gateways to represent the executions.

Given a process P, its normal $n(P)$ can be obtained as follows:

1. Repeatedly replace each penning-parallel gateway g by an opening-exclusive gateway e. For each $v \in suc(e)$, construct a new penning-parallel gateway g' with $pre(g') = v$, $suc(g') = suc(v) \cup suc(e) \setminus \{v\}$ and then make $suc(v) = g'$.
2. Remove all the edges from an opening- to a closing-parallel gateway.
3. If an opening-gateway has only one successor, remove the gateway
4. If an closing-gateway has only one predecessor, remove the gateway

In step 1 direct successors of parallel gateways are "pulled" out of the gateway. Here a loop block is considered as a single successor. In this procedure, a parallel gateway with n successors is transformed into n parallel gateways with n successors but one of the successive sequence is shortened by one successor. Due to the finite length of these sequences, this replacement always terminates. Step 2 then reduces the number of successors for these remaining parallel gateways by removing "empty" edges. Step 3 and 4 finally remove the gateways. When a gateway is removed, its predecessors and successors should be directly connected.

It's obvious that this normalisation will always result in a normal process. An example of normalisation of Fig.1b and Fig.1d can be seen in Fig.2.

The size of $n(P)$ can be exponentially large w.r.t. P in worst case: suppose P contains only a pair of parallel gateways with n sequences of one activity, then $n(P)$ will contains a pair of exclusive gateways with $n!$ sequences of n activities.

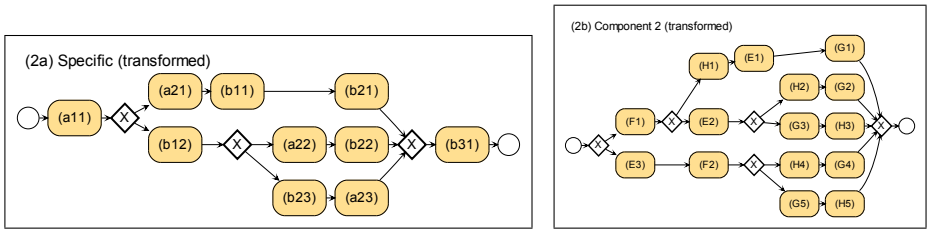

Fig. 2. Transformation to execution diagrams for Fig. 1b and d

In normalisation, some activities will be duplicated in the process. These duplications have different predecessors (successors). We distinguish them by an additional numerical subscript. We depict such a transformed process $n(P)$ with distinguished activities by $P^\star$. Obviously, $ES_{P^\star}$ is the same as $ES_{n(P)}$ after replacing all the distinguished activities by their original names. Thus, relation between two execution sets can be characterised by the following theorem:

Theorem 1. *Given two processes* $P = \langle \mathrm{E}_P, \mathrm{V}_P \rangle$ *and* $Q = \langle \mathrm{E}_Q, \mathrm{V}_Q \rangle$, $ES_Q \subseteq ES_P$ iff $\forall a_i \in \mathrm{A}_{Q^\star}$, *there exists some* $a_j \in \mathrm{A}_{P^\star}$ *such that* $PS_{Q^\star}(a_i) \subseteq PS_{P^\star}(a_j)$ *and* $SS_{Q^\star}(a_i) \subseteq SS_{P^\star}(a_j)$.

Proof. (1) For the $\rightarrow$ direction the lhs $ES_Q \subseteq ES_P$ holds. We demonstrate the subsumption for PS. For an arbitrary activity $a_i \in \mathrm{A}_{Q^\star}$, the activity $a_i{}'$ is the corresponding activity before normalisation (i.e. without additional subscripts).

The activity $a_j{}'$ is the originator or ground activity of $a_i{}'$ in P after renaming and $a_j \in \mathrm{A}_{P^\star}$ is the the corresponding activity after normalization of P. From the prerequisite it directly follows that the predecessors of $a_i \in \mathrm{A}_{Q^\star}$ are predecessors of a_j in $Q^\star$. The subsumption of the successor set is demonstrated likewise. (2) To prove the other direction we assume that the rhs holds. Consider an execution $s \in ES_Q$ we demonstrate that $s \in ES_P$. For each activity $a_i{}'$ of an arbitrary execution $s \in ES_Q$ the corresponding activity $a_i \in \mathrm{A}_{Q^\star}$ is received after normalization. From the rhs it follows that there exists an activity $a_j \in \mathrm{A}_{P^\star}$ so that each predecessor of a_i is also a predecessor of a_j in $P^\star$ and likewise for the successors of a_i. After activity renaming and demonstrating for all activities of each execution of ES_Q the inclusion of the lhs follows.

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 its process model and the predecessor and successor sets:

- **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 connect their predecessors and successors directly, the resulting $ES_{Q'}$ will be the same as the reduced ES_Q with respect to P.
- **Renaming** can also be directly performed on the process diagram, i.e. $ES_P[a \to A] = ES_{P[a \to A]}$. Thus, the renaming can be performed on the predecessor and successor sets as well, i.e. $PS_P(x)[a \to A] = PS_{P[a \to A]}(x)$ $(SS_P(x)[a \to A] = SS_{P[a \to A]}(x))$.
- **Decomposition** can be done on the predecessor and successor sets as well. Theorem 1 shows that the subsumption of execution sets can be reduced to subsumption of predecessor and successor sets. Decomposition means, an activity x can go from not only predecessors of x, but also another appearance of x, and can go to not only successors of x, but also another appearance of x. Any sequence of x in the execution will be decomposed.

Thus, 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 a process P and obtain the predecessor and successor sets from $P^\star$, then encode **Renaming** and **Decomposition** into ontology 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/successors relations are described by two roles *from* and *to*, respectively.

On instance level, these two roles should be inverse role of each other. However this is not necessary in our solution. Composition of activities in horizontal refinement is described by role *compose*. Grounding of activities in vertical refinement is described by role *groundedTo*. To facilitate the ontology construction, four operators are defined for pre- and post- refinement process:

Definition 1. *: Given S a predecessors or successors set, we define four operators for translations as follows:*
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
$Disjoint(x|x \in \mathrm{A}_P \cup \mathrm{A}_Q)$ *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
$Disjoint(x|x \in \mathrm{A}_P \cup \mathrm{A}_Q)$ *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 the $\rightarrow$ 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) The $\leftarrow$ direction can be proved similarly by contraposition.

Now we can represent horizontal and vertical refinements by ontologies:

Horizontal Refinement For conciseness of presentation, we always have a pre-refinement process P and a post-refinement process Q and we refine one activity z of P in this step. z may have multiple appearances z_j in $P^\star$. For each z_j we define $component\_z_j \equiv \exists compose.z_j$. Simultaneous refinement of multiple activities can be done in a similar manner of single refinement. Then we construct an ontology $\mathcal{O}_{P \rightarrow Q}$ with following axioms:

1. for each activity $a_i \in \mathrm{A}_{Q^\star}$ and $hori(a) = z$
   $a_i \sqsubseteq \bigsqcup \exists compose.z_j$
   These axioms represent the composition of activities with concept subsumption, which realise **Renaming** in horizontal refinement. For example, $b_{31} \sqsubseteq \exists compose.B$ and $a_{11} \sqsubseteq \exists compose.A$.
2. for each $a_i \in \mathrm{A}_{Q^\star}$ where a is not refined from z
   $a_i \sqsubseteq \mathbf{Pr}_{from}(PS_{P^\star}(a_i))[z_j \rightarrow component\_z_j]$,
   $a_i \sqsubseteq \mathbf{Pr}_{to}(SS_{P^\star}(a_i))[z_j \rightarrow componennt\_z_j]$,
   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 $\bigsqcup component\_z_j$, we replace the appearance of each z_j by corresponding $component\_z_j$. For example, $Start \sqsubseteq \forall to.component\_A$.
3. for each $z_j \in \mathrm{A}_{P^\star}$,
   $component\_z_j \sqsubseteq \mathbf{Pr}_{from}(PS_{P^\star}(z_j) \cup \{component\_z_j\})[z_j \rightarrow componennt\_z_j]$,
   $component\_z_j \sqsubseteq \mathbf{Pr}_{to}(SS_{P^\star}(z_j) \cup \{component\_z_j\})[z_j \rightarrow componennt\_z_j]$,
   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 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, $component\_A \sqsubseteq \forall from.(Start \sqcup component\_A)$.
4. for each $a_i \in \mathrm{A}_{Q^\star}$,
   $a_i \sqsubseteq \mathbf{Ps}_{from}(PS_{Q^\star}(a_i))$,
   $a_i \sqsubseteq \mathbf{Ps}_{to}(SS_{Q^\star}(a_i))$,
   These axioms represent the predecessor and successor sets of all the activities in the post-refinement process. For example, $a_{22} \sqsubseteq \exists from.b_{12}$, $b_{23} \sqsubseteq \exists to.a_{23}$.
5. $Disjoint(a_i | a_i \in Q$ and $Hori(a) = z)$
   These axioms represent the uniqueness of all the sibling activities refined from the same z_j. For example, $Disjoint(a_{11}, a_{21}, a_{22}, a_{23})$
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 the above axioms, ontology $\mathcal{O}_{P \rightarrow Q}$ 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, suppose we have principle behaviour model P and a concrete process model Q, which has already been reduced w.r.t P to eliminate ungrounded activities. Any activity in Q can be grounded to some activity in P. Thus, after reduction, $\forall a \in \mathrm{A}_P, \exists b \in \mathrm{A}_Q$ that b is grounded to a, and vice versa. Therefore for each $x_j \in \mathrm{A}_{P^\star}$, we define $grounded\_x_j \equiv \exists groundedTo.x_j$.Then we construct an ontology $\mathcal{O}_{P \rightarrow Q}$ with following axioms:

1. for each activity $a_i \in \mathrm{A}_{Q^\star}$ and $vert(a) = x$
   $a_i \sqsubseteq \bigsqcup \exists groundedTo.x_j$

These axioms represent the grounding of activities by concept subsumption, which realise the **Renaming** in vertical refinement. For example, $a_{11} \sqsubseteq \exists groundedTo.E$, $b_{11} \sqsubseteq \exists groundedTo.F$.

2. for each $a_i \in \mathrm{A}_{P^\star}$
   $grounded\_a_i \sqsubseteq \mathbf{Pr}_{from}(PS_{P^\star}(a_i))[x_j \rightarrow grounded\_x_j]$,
   $grounded\_a_i \sqsubseteq \mathbf{Pr}_{to}(SS_{P^\star}(a))[x_j \rightarrow grounded\_x_j]$,
   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 $x_j \in \mathrm{A}_{P^\star}$ 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 $Q^\star$.
3. for each $a_i \in \mathrm{A}_{Q^\star}$,
   $a_i \sqsubseteq \mathbf{Ps}_{from}(PS_{Q^\star}(a_i))$,
   $a_i \sqsubseteq \mathbf{Ps}_{to}(SS_{Q^\star}(a_i))$,
   These axioms represent the predecessor and successor sets of all the activities in the post-refinement process. Notice that the ungrounded activities have been removed from the process.
4. for each $x \in \mathrm{A}_P$,
   $Disjoint(a_i | a_i \in Q^\star$ and $vert(a) = x)$
   These axioms represent the uniqueness of all the sibling activities refined from the same x.
5. $Disjoint(Start, End,$ all the $grounded\_x_j)$.
   This axiom represents the uniqueness of all the activities before refinement. For example, $Disjoint(Start, End, grounded\_C, grounded\_D)$.

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

In both horizontal and vertical refinement, the number of axioms are linear w.r.t. the size of $P^\star$ and $Q^\star$. The language is $\mathcal{ALC}$.

3.3 Concept satisfiability checking

In ontology $\mathcal{O}_{P \rightarrow Q}$, all the activities in $Q^\star$ satisfy the ordering relations in $P^\star$ by satisfying the universal restrictions and satisfy the ordering relations in $Q^\star$ by satisfying existential restrictions. Given the uniqueness of concepts, the inconsistency between $P^\star$ and $Q^\star$ will lead to unsatisfiability of particular concepts. The relation between the ontology $\mathcal{O}_{P \rightarrow Q}$ and the validity of the refinement from P to Q is characterised by the following theorems:

Theorem 3. *An execution path containing activity a in Q is invalid in the refinement from P to Q,* iff *there is some $a_i \in Q^\star$ such that $\mathcal{O}_{P \rightarrow Q} \models a_i \sqsubseteq \bot$.*

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 2 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 2 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.

we check the satisfiability of the concepts to validate the process refinement. Every unsatisfiable concept is either an invalid refinement or related to an invalid refinement.

With the help of reasoning, we can easily see that Fig.1b is an invalid horizontal refinement w.r.t. Fig.1a: $a_{22} \sqsubseteq \exists from.b_{12}$, also $a_{22} \sqsubseteq \exists compose.A$ thus $a_{22} \sqsubseteq \forall from.(Start \sqcup component\_A)$. However, b_{12} disjoints with both $Start$ and $component\_A$ therefore a_{22} is unsatisfiable. Similarly, we can detect that b_{12}, b_{23} and a_{23} are unsatisfiable. This implies the invalid routes in Fig.2a and further the invalid refinement of Fig.1b. Also, the vertical refinement w.r.t. Fig.1d is wrong while the vertical refinement w.r.t. Fig.1c is correct.

According to the underlying logic, reasoning complexity is ExpTime.

Helped by our analysis, the process architect remodels their process (Fig. 3). Now, the execution set of Fig. 3 is $\{[\mathsf{a_1a_2b_1b_2b_3}], [\mathsf{a_1a_2b_2b_1b_3}], [\mathsf{a_1a_2b_1b_3b_2}]\}$. Renaming of Fig. 3's execution set with respect to Fig. 1a yields $\{[\mathsf{AABBB}]\}$. After decomposition, we conclude that Fig. 3 correctly horizontally refines the process in Fig. 1a because $\{[\mathsf{AB}]\} \supseteq \{[\mathsf{AB}]\}$. As for validating vertical refinement with the component models, renaming yields $\{[\mathsf{EFGHD}], [\mathsf{EFHGD}], [\mathsf{EFGDH}]\}$. After reduction with respect to Fig. 1c and Fig. 1d, we conclude that Fig. 3 correctly grounds on Fig. 1c and Fig. 1d because $\{[\mathsf{C}], [\mathsf{D}], [\mathsf{CC}], [\mathsf{CD}], [\mathsf{DC}], [\mathsf{DD}], \ldots\} \supseteq \{[\mathsf{D}]\}$ and $\{[\mathsf{EFGH}], [\mathsf{EFHG}], [\mathsf{FHEG}], [\mathsf{FEGH}], [\mathsf{FEHG}]\} \supseteq \{[\mathsf{EFGH}], [\mathsf{EFHG}]\}$.

4 Evaluation

We have implemented the transformation of BPMN process models and refinement information to OWL-DL ontology. In addition to the transformation, we implemented a generator which creates random, arbitrarily complex refinement scenarios. Flow correctness is ensured by constructing the process models out of block-wise patterns that can be nested. The generator is parameterized by

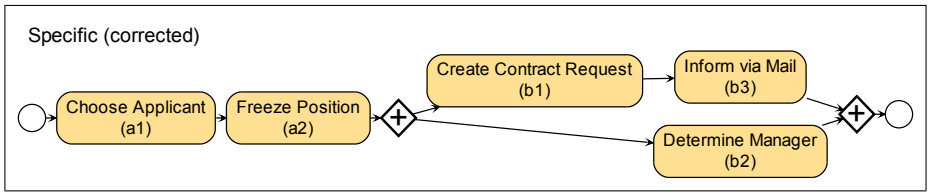

Fig. 3. Adapted specific process for correct refinement

the maximum branching factor B, the maximum length L of a pattern instance, the maximum depth of nesting N, and by the probability for loops, parallel, or exclusive flow. Most realistic appearing process diagrams were created with $B = 3$, $L = 6$, $N = 3$, and with a mixture of loops, parallelism, and exclusive flow to the ratio of $2 : 1 : 2$.

With the given parameters, we generated 1239 refinement scenarios (197 correct, 1042 wrong) with the average and maximum number of activities in the generic and refined models printed on the left-hand side below.

	Generic Activities	Specific Activities	Total Activities	Transf. Time	OWL-DL Axioms	Reasoning Time
Average	5.79	17.4	23.2	4ms	154	2.8s
Maximum	30	53	69	0.4s	1159	3.4min

The generated scenarios were used to evaluate the refinement analysis on a laptop with a 2 GHz dual core CPU, 2 GB of RAM using Java v1.6 and Pellet 2.0.0. Two factors contribute to the overall complexity of the analysis:

1. **Transformation to OWL-DL.** As we pointed out earlier, theoretically, the complexity of the transformation can be exponential in the worst case. However, our experiments show that in many practical cases, the size of the OWL-DL knowledge base—measured by the number of axioms—remains relatively small. In particular, for 80% (90%) of the scenarios, the number of axioms was below 220 (400) (see Fig. 4). Some unusual nesting of parallel flow causes the exceptions in the diagram that have a higher number of generated axioms. Remarkably, the appearance of such cases seems to uniquely distribute over the scenarios independently of the size of the original processes due to the artificial nature of the generated scenarios.
2. **OWL-DL Reasoning.** The theoretical complexity of OWL-DL reasoning is exponential as well. However, our evaluation runs in Fig. 5 suggest that for the practical cases evaluated, reasoning time grows less than exponentially (less than a straight line on a logarithmic scale) compared to the number of axioms in the OWL-DL knowledge base. We separately plot the reasoning times of the correct and wrong refinements because classifying a consistent knowledge base is more expensive in general.

When comparing absolute times, reasoning consumes about two orders of magnitude more time than transformation as can be seen from the right-hand

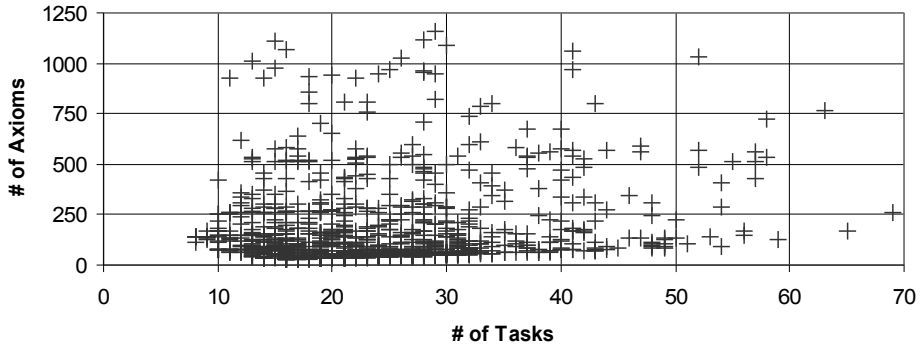

Fig. 4. Axioms for activities

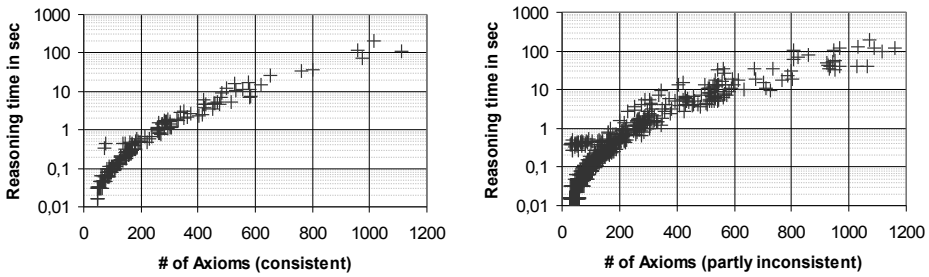

Fig. 5. Reasoning time for axioms on logarithmic scale

side of the table above. This determines our future research to seek improvements in the reasoning rather than in the transformation.

As for the complete run time, the above table indicates that the refinement analysis of an average scenario—with a generic process of about 6 activities and a refining process of about 17 activities—would take about 3 seconds. We consider this a simpler, yet realistic problem size.

In one of the larger evaluated scenarios, 15 generic activities were refined to 48 specific activities (for comparison: our running example contains 5 specific activities). The 63 activities in total (= 15+48) were transformed to 402 activities due to many parallel flows in that scenario. Analysis of the 765 generated axioms was performed in 18 seconds.

In the most complicated scenario of our evaluation, where a large knowledge base had to be constructed due to the heavy use of parallel gateways, total analysis time remained below 4 minutes. Although this is definitely too much for providing a real-time refinement check to process modelers, analysis took less than 1 second for 80% ($\leq$ 220 axioms) and less than 10 seconds for 90% ($\leq$ 400 axioms) of the examined practical cases. Compared to the manual efforts a human is required today, our approach provides a significant improvement. Furthermore, the check performed by our approach is—in contrast to the manual

approach—guaranteed to be correct and thus helps to avoid costly follow-up process design errors.

5 Related Works and Conclusion

There are many existing works related to our study. Some of them [17, 14] come from the business process modelling and management community which investigated process property checking with model checkers.

Researchers in system transition and communication systems [13, 11, 10, 12] also developed behaviour algebra to analyse the bisimulation, i.e. matching between processes. In some of the works, execution set semantics are also applied [18]. However, these models do not validate refinement with activity compositions.

Other models use mathematical formalisms to describe concurrent system behaviour. [3] describes concurrent system behaviour with operational semantics and denotational semantics. But the analyzed equivalence between process models does not distinguish between deterministic and non-deterministic choices.

Semantic web community contribute to this topic by providing first semantic annotations for process models such as service behaviour and interaction [4, 16, 15] and later automatic process generation tools [7]. However, these approaches do neither consider process refinement nor a DL based validation of relationships.

In [8] actions and services, which are a composition of actions are described in DL. Actions contain pre- and post-conditions. The focus is on a generic description of service functionality. As inference problems, the realizability of a service, subsumption relation between services and service effects checking is analyzed. Services are described similarly with DL in [2]. The reasoning tasks are checking of pre- and post-conditions of services. The main focus of this work is the reasoning complexity.

The DL $\mathcal{DLR}$ is extended with temporal operators in [1] for temporal conceptual modelling. In this extension, query containment for specified (temporal) properties is analyzed. In [9] the DL $\mathcal{ALC}$ is extended with the temporal logics LTL and CTL. Still, neither of them considers process modelling and refinements.

Our contribution is this paper includes:

1. Devising a general approach to represent and reason with process models containing parallel and exclusive gateways;
2. Applying graph-based topological approach with DL reasoning to provide automatic solution of process refinement checking;
3. Implementing and evaluating a prototype that performs process transformation and refinement checking as proposed.

In the future, there are several potential extension of this work. We will continue our implementation and evaluation to support larger and more complex process models. We will also try to extend the process representation with more expressive power. Another interesting topic is whether the process transformation itself can be automatically inferred by reasoning. We also want to integrate our refinement representation with other business process modelling ontologies.

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. M. Hepp, F. Leymann, C. Bussler, J. Domingue, A. Wahler, and D. Fensel. Semantic business process management: Using semantic web services for business process management. *Proc. of the IEEE ICEBE*, 2005.
6. M. Hepp and D. Roman. An Ontology Framework for Semantic Business Process Management. In *Proc. of 8th Internalional Conference Wirtschaftsinformatik*, 20007.
7. Joerg Hoffmann, Ingo Weber, T. Kaczmarek James Scicluna, and Anupriya Ankolekar. Combining Scalability and Expressivity in the Automatic Composition of Semantic Web Services. In *Proceedings of the 8th International Conference on Web Engineering (ICWE 2008)*, 7 2008.
8. C. Lutz and U. Sattler. A Proposal for Describing Services with DLs. In *Proceedings of the 2002 International Workshop on Description Logics*, 2002.
9. 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.
10. R. Milner. *A Calculus of Communicating Systems*. Springer LNCS, 1980.
11. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
12. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes. *Information and Computation*, pages 41–77, 1992.
13. Davide Sangiorgi. Bisimulation for Higher-Order Process Calculi. *Information and Computation*, 131:141–178, 1996.
14. WMP van der Aalst, HT de Beer, and BF van Dongen. Process Mining and Verification of Properties: An Approach based on Temporal Logic. *LNCS*, 3761:130–147, 2005.
15. I. Weber, J. Hoffmann, and J. Mendling. Semantic Business Process Validation. In *Proc. of International workshop on Semantic Business Process Management*, 2008.
16. 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.
17. P.Y.H. Wong and J. Gibbons. A process-algebraic approach to workflow specification and refinement. *Lecture Notes in Computer Science*, 4829:51, 2007.
18. 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.