Process Refinement Validation and Explanation with Ontology Reasoning

Yuan Ren[1], Gerd Gröner[2], Jens Lemcke[3], Tirdad Rahmani[3], Andreas Friesen[3], Yuting Zhao[1], Jeff Z. Pan[1], and Steffen Staab[4]

[1] University of Aberdeen
[2] PALUNO – University of Duisburg-Essen
[3] SAP AG
[4] University of Koblenz-Landau

Abstract. In process engineering, processes can be refined from simple ones to more and more complex ones with decomposition and restructuring of activities. The validation of these refinements and the explanation of invalid refinements are non-trivial tasks. This paper formally defines process refinement validation based on the execution set semantics and presents a suite of refinement reduction techniques and an ontological representation of process refinement to enable reasoning for the validation and explanation of process refinement. Results show that it significantly improves efficiency, quality and productivity of process engineering.

1 Introduction

It is germane in process management to represent processes at different levels of abstraction, ranging from abstract processes (coarse description) to specific processes (fine-grained characterisation). Due to the different levels of abstractions, it is not obvious to determine whether or not a refined process reflects the intended behaviour of the original process. This makes the validation of refinements and the explanation of sources for invalidity crucial issues. Manual validation is usually error-prone, time-consuming and increases the cost of process engineering. Existing (semi-) automatic methods still limit the flexibility in process refinement. In this paper, we make the following contributions to the automatic validation of process refinement.[1]

- Based on the classic execution set semantics, we propose a formal and intuitive semantics of process refinement (Sec. 2).
- Based upon the above semantics, we propose a novel approach to automatically validate and explain process refinements (Sec. 3) by combining graph-based transformation and ontology reasoning.
- We implemented our approach and conducted evaluations in terms of performance and usefulness (Sec. 4). Experiments show that realistic refinement scenarios can be validated below one minute and average-sized problems in a split second. This significantly improves the quality and productivity of process engineering.

[1] Detailed proofs of all theorems can be found in our online technical report: `http://homepages.abdn.ac.uk/jeff.z.pan/pages/pub/ProcessRefinement.pdf`

S. Basu et al. (Eds.): ICSOC 2013, LNCS 8274, pp. 515–523, 2013.

2 Problem Description

A *process* (or *process model*) is a directed graph $\mathrm{P} = \langle \mathrm{V}, \mathrm{E} \rangle$ without multiple edges between two vertices. Vertices (V) include activities and gateways ($\mathrm{A}, \mathrm{G} \subseteq \mathrm{V}$). The start and end events ($v^{\mathrm{S}}, v^{\mathrm{E}} \in \mathrm{A}$) are two special activities, e.g., process P_1 in Fig. 1 consists of two activities A and B between the start and end events. No activity is allowed twice in a process model.

A gateway is either opening or closing ($\mathrm{G}^{\mathrm{O}}, \mathrm{G}^{\mathrm{C}} \subseteq \mathrm{G}$), and either exclusive or parallel ($\mathrm{G}^{\circledast}, \mathrm{G}^{\oplus} \subseteq \mathrm{G}$). Process P_3 contains parallel gateways and exclusive gateways. Exclusive gateways can be used to construct loops, e.g., in P_1, A and B can be repeatedly executed. The set of edges (E) is a binary relation on V. For each $v_1 \in \mathrm{V}$, we know its *direct predecessors (successors)* $\mathrm{pre}(v_1) := \{v_2 \in \mathrm{V} \mid (v_2, v_1) \in \mathrm{E}\}$ ($\mathrm{suc}(v_1) := \{v_3 \in \mathrm{V} \mid (v_1, v_3) \in \mathrm{E}\}$). Given a valid process model $\mathrm{P} = \langle \mathrm{V}, \mathrm{E} \rangle$ with $|\mathrm{pre}(v^{\mathrm{S}})| = |\mathrm{suc}(v^{\mathrm{E}})| = 0$, $|\mathrm{suc}(v^{\mathrm{S}})| = |\mathrm{pre}(v^{\mathrm{E}})| = 1$; $\forall o \in \mathrm{G}^{\mathrm{O}}$ $(c \in \mathrm{G}^{\mathrm{C}})$, $|\mathrm{pre}(o)|(= |\mathrm{suc}(c)|) = 1$; $\forall a \in \mathrm{A} \setminus \{v^{\mathrm{S}}, v^{\mathrm{E}}\}$, $|\mathrm{pre}(a)| = |\mathrm{suc}(a)| = 1$, we define gateway-free predecessor as $PS(v_1) := \{v_2 \in \mathrm{A} \setminus \{v^{\mathrm{S}}\} \mid v_2 \in \mathrm{pre}(v_1)$ or $\exists u \in \mathrm{G}$, $u \in \mathrm{pre}(v_1)$ and $v_2 \in PS(u)\}$ and successor as $SS(v_1) := \{v_3 \in \mathrm{A} \setminus \{v^{\mathrm{E}}\} \mid v_3 \in \mathrm{suc}(v_1)$ or $\exists u \in \mathrm{G}$, $u \in \mathrm{suc}(v_1)$ and $v_3 \in SS(u)\}$. These two definitions make gateways "transparent", e.g., in P_2, $SS(A_1) = \{A_2, B_1\}$. In the following, we refer to elements of PS (SS) as predecessors (successors) for short.

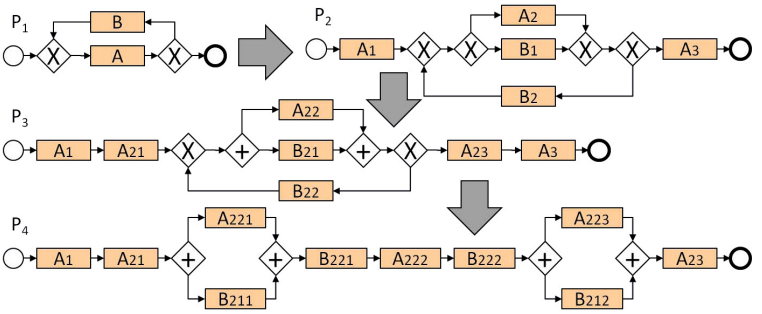

Fig. 1. A chain of process refinements

A *process refinement* is a transformation from an abstract process into a more specific one. An example of a chain of several process refinements is depicted in Fig. 1, in which P_2 refines P_1 by decomposing activity A into A_1, A_2 and A_3, B into B_1 and B_2. P_2 is further refined by P_3, in which A_2 is further decomposed into A_{21}, A_{22} and A_{23}, B_2 into B_{21} and B_{22} and so on.

The semantics of a process is based on its *executions*. An execution is a *proper* sequence of activities $a_i \in \mathrm{A}$: $[a_1 a_2 \ldots a_n]$. It starts from one of the successors of v^{S} and continues with subsequent activities. The ordering relations among activities must be obeyed, i.e., an activity a must be appended to the sequence before all *SS(a)* and after all *PS(a)*. When it comes to an exclusive gateway ($\circledast$), a proper sequence can go through exactly one exclusive branch. For example in

P_1, after appending A, a sequence can be either terminated by v^{E}, or continued by B. When it comes to a parallel gateway ($\diamondsuit$), a proper sequence must go through all parallel branches. For example in P_3, after appending A_{21}, a sequence must append both A_{22} and B_{21} before making a choice between B_{22} and A_{23}. The ordering between A_{22} and B_{21} can be arbitrary. The result is a proper sequence of activities—an execution:

Definition 1 (Execution Set). *The execution set of a process P, denoted by ES_P, is the (possibly infinite) set of all executions of P.*

For example, ES_{P_1} for process P_1 in Fig. 1 is $\{[A], [ABA], [ABABA], \dots\}$. Process P_3 contains parallel gateways to express that some activities can be executed in any order: $ES_{P_3} = \{[A_1A_{21}A_{22}B_{21}A_{23}A_3], [A_1A_{21}B_{21}A_{22}A_{23}A_3], \dots\}$.

The MIT business process handbook [1] characterises the behaviour of a process in terms of its execution set semantics, and the refinement is specified by the comparison of the execution sets of an abstract and a specific process. A process P subsumes another process Q under the maximal execution set semantics *iff* $ES_Q \subseteq ES_P$. However, architects might use different activity names in abstract and specific processes. Thus, the process architect has to declare which activities of the specific process refine which activity of the abstract process. This is denoted by the *orig*-function, e.g., $orig(A_1) = orig(A_2) = A.$, The *orig*-function is extended to executions and execution sets, e.g., applying this to P_2 yields $orig(ES_{P_2}) = orig(\{[A_1A_2A_3], [A_1A_2B_2A_2A_3], \dots\}) = \{[AAA], [AABAA], \dots\}$. Furthermore, an activity of the abstract process might be *decomposed* into multiple activities in the specific process. Even if we discard the different names, the specific activities still outnumber their origins. For example, process P, consisting of a single activity A, is refined into a process Q with consecutive subactivities $A_1, \dots, A_n$. Intuitively, Q is a valid refinement of P but the execution of P has length 1 and the execution of Q has length n. To resolve this, we define the *decomposable process* P^D of P that is constructed from P by constructing a loop around every activity of P, except the start and end event.

3 Validation and Explanation with Ontologies

The definition of valid refinement is intuitive without parallel gateways since all orderings are explicitly stated. Accordingly, we first present the validation of parallel-free process refinements. Afterwards, we extend our approach to incorporate parallel gateways.

3.1 Validating Parallel-Free Process Refinement

For parallel-free refinements, we use (Description Logics) ontologies and reasoning to validate and explain refinements. A Description Logics (DL) ontology consists of a terminology box (TBox) and an assertion box (ABox). The TBox describes the schematic knowledge with concepts and roles. In this paper, the ontology will be built in the DL fragment $\mathcal{ALC}$. Concepts are inductively defined

by the following constructs: $\top \mid \bot \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists r.C \mid \forall r.C$, in which $\top$ denotes the universal set of the domain, $\bot$ denotes the empty set, A is a named concept, C and D are arbitrary concept expressions. R and S are roles. $\neg C$ is the negation of C, $\sqcap$ and $\sqcup$ the conjunction and disjunction. $\exists r.C$ and $\forall r.C$ represent the set of individuals who have an r relation to some instance of C, or r relations only to instances of C. The subsumption between two concepts C and D is depicted as $C \sqsubseteq D$. Two concepts are disjoint if $C \sqcap D \sqsubseteq \bot$. We write $Disjoint(C_1, C_2, \ldots, C_n)$ to denote that any two C_i, C_j $(1 \leq i, j \leq n, i \neq j)$ are mutually disjoint with each other. If an axiom α can be inferred from an ontology $\mathcal{O}$, we say $\mathcal{O}$ entails α, denoted by $\mathcal{O} \models \alpha$. We mainly use the subsumption checking reasoning service, i.e., checking if $\mathcal{O} \models C \sqsubseteq D$.

The abstract process restricts the set of "allowed" predecessors and successors, while the specific process states the "existing" predecessors and successors after the refinement. We restrict predecessor and successor relations in the abstract process by universal restrictions ($\forall$), and existential quantifications ($\exists$) describe predecessors and successors of activities from the specific process. For both process models, we use the same roles *to* and $from$ for successor and predecessor relationships. Formally, the ontology $\mathcal{O}$ is built as follows:

Definition 2 (Refinement Ontology). *Let S be a set of predecessors or successors, respectively, 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* $\boldsymbol{Ps}_{to}(S) = \bigsqcap_{y \in S} \exists to.y$

For conciseness, we always have one abstract process P and one specific process Q. In order to detect invalid refined activities, we introduce the concept *Invalid*. Then, we construct an ontology $\mathcal{O}_{P \to Q}$ with the following patterns. The refinement from P_1 to P_2 in Fig. 1 is used as an example.

1. For each activity $X \in \mathrm{A}_Q$ with $orig(X) = Z$, we use $X \sqsubseteq Z$ to represent the composition of activities, which covers the *activity origin*.
2. For each activity $X \in \mathrm{A}_P$, we use $X \sqsubseteq Invalid \sqcup \mathbf{Pr}_{from}(PS_{P^D}(X))$, $X \sqsubseteq Invalid \sqcup \mathbf{Pr}_{to}(SS_{P^D}(X))$ to describe the activities in the pre-refinement process. Due to possible decompositions of activities, we use the decomposable process to characterise the predecessor and successor sets, e.g., $A \sqsubseteq Invalid \sqcup \forall from.(Start \sqcup A \sqcup B)$, $A \sqsubseteq Invalid \sqcup \forall to.(End \sqcup B \sqcup A)$, $B \sqsubseteq Invalid \sqcup \forall from.(B \sqcup A)$, $B \sqsubseteq Invalid \sqcup \forall to.(A \sqcup B)$. The pre-refinement process restricts allowed predecessor and successor activities. Thus, the $Invalid$ concept is added as an alternative, implying that if any component of X does not satisfy the ordering constraints of X, it will become $Invalid$.
3. For each activity $X \in \mathrm{A}_Q$, we use $X \sqsubseteq \mathbf{Ps}_{from}(PS_Q(X))$, $X \sqsubseteq \mathbf{Ps}_{to}(SS_Q(X))$ to represent predecessor and successor sets of it in the specific process.
4. $Disjoint(X | X \in \mathrm{A}_Q)$, $orig(X) = Z$ (for $X \in \mathrm{A}_Q$). These axioms represent the uniqueness of all activities with the same origin, e.g., $Disjoint(A_1, A_2, A_3)$.

5. $Disjoint(X|X \in \mathrm{A}_P)$. This axiom represents the uniqueness of all the activities in the abstract process, e.g., $Disjoint(Start, End, A, B)$.

With the above axioms, ontology $\mathcal{O}_{P\to Q}$ is a representation of the refinement from P to Q. All executions of Q can be represented by some existential restrictions ($\exists$). Given the subsumption of activities, these $\exists$ chains must satisfy the universal restrictions ($\forall$) in $\mathcal{O}_{P\to Q}$ to fulfill the executions of P^D. Due to the uniqueness of concepts, an invalid refinement between P^D and Q will make invalid refined activities to be subsumed by $Invalid$. This helps to pinpoint the source of an invalid refinement.

Theorem 1. *For any parallel-free refinement from P to Q, the refinement is invalid due to activity $A \in \mathrm{A}_Q$,* iff $\mathcal{O}_{P\to Q} \models A \sqsubseteq Invalid$.

3.2 Extending Processes with Parallel Gateways

The presence of parallel gateways requires some pre-processing steps on the abstract and/or specific processes before building the refinement ontology.

If the *specific process* contains parallel gateways, e.g., the refinement from P_2 to P_3 in Fig. 1, we observe that parallel branches implicitly describe different possible executions among activities in sibling branches. E.g., in P_3, there are two parallel branches, each of them contains one activity (A_{22} or B_{21}). The implicit executions of the parallel sibling activities A_{22} and B_{21} are $[A_{22}B_{21}]$ and $[B_{21}A_{22}]$, i.e., either activity A_{22} is executed before B_{21} or B_{21} before A_{22}. For the validation, we have to take all these implicit executions of parallel branches into account. To remedy this, we replace all parallel gateways with exclusive gateways and connect the input and output of all previously parallel activities (with the help of exclusive gateways). Fig. 2 illustrates a replacement of P_3.

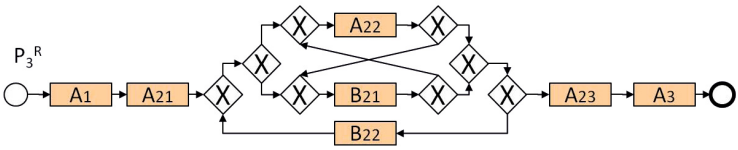

Fig. 2. P_3^R: Replaced process P_3

If the *abstract process* contains parallel gateways, as in the refinement from P_3 to P_4, we observe: (i) Activities A_{22} and B_{21} of the abstract process P_3 are in parallel. According to the execution set semantics, activities A_{22} and B_{21} can be executed in any order. (ii) The decomposition principle allows for an infinite repetition of activities. Thus, in the specific process P_4, this implies that the ordering relations between decompositions of A_{22} and B_{21}, e.g., A_{221}, B_{211}, etc. in P_4, do not affect the validity of the refinement from P_3 to P_4. To remedy this, the sibling parallel activities A_{22} and B_{21} can be regarded as "transparent" to each other in the refinement checking. Activities of parallel sibling branches

(e.g., A_{22} or B_{21}) are removed in the abstract process, and their corresponding decomposed activities are removed in the specific process, respectively. Thus, execution relations of activities in parallel sibling branches are neglected in the abstract process. We refer to this reduction as *parallel branch break-down.*

4 Evaluation

The technical performance of our approach is influenced by the run-time of DL reasoning. We implemented a generator that simulates arbitrarily complex refinement scenarios with different characteristics. We used a standard laptop with a 2.67 GHz dual core, four-threads CPU with 8 GB RAM using Java 1.6 and TrOWL v0.5.1 (`http://trowl.eu`). Fig. 3 suggests that the reasoning time of arbitrary parallel-free refinements (dotted graph in the left figure) grows less than exponentially (less than a straight line on a logarithmic scale) compared to the number of activities. The parallel branch break-down contributes most to performance degradation. We generated sequences where each sequential step consists of two parallel activities. As the combination of all branches has to be considered separately, the left figure shows exponentially growing run-times.

To estimate the performance in practice, we generated processes with 25% parallel flow, 50% exclusive flow, and 25% loops because that ratio appeared most natural. The validation run-times are plotted on the right of Fig. 3. In [2], IBM examined 735 industrial business processes from different domains. The processes contained 17 activities on average. The maximum was 118. Thus, a realistic refinement scenario would contain about 34 activities on average and 236 at maximum. As can be seen from the figure, the corresponding run-times of our approach would be 0.062s for the average and about 40s for the largest process.

In addition to the technical performance, we also assessed the business value (productivity and quality) of refinement validation. To quantify the (1) productivity and (2) quality improvements through automatic refinement validation, we conducted a multiple choice test with 13 experts from model-driven software development. The test consists of two sets of 20 questions. Each question refers to three different processes P_1, P_2, and P_3 from [3], where P_1 refines P_2 and P_2 refines P_3. Each question has between two and four answer options where none or multiple answers can be correct. For the one set of 20 questions, denoted by S for "support", the result of the refinement validation is highlighted in the process models. The other set, denoted by N for "no support", has to be answered without support. We measured the number of correct and wrong answers per set (C_S, W_S, C_N, W_N) and the times for each set (t_S, t_N). The whole test took about 1.5 hours. We calculate each person's quality of answers as $Q_x = C_x/(C_x + W_x)$ and productivity as $P_x = C_x/t_x$, where $x \in \{S, N\}$. In order to abstract from personal work styles and experiences, we calculate the per-person improvement in quality as $QI = \frac{Q_S}{Q_N} - 1$ and in productivity as $PI = \frac{P_S}{P_N} - 1$. Our test reveals an average per-person improvement in quality of $\overline{QI} = 70\%$ and in productivity of $\overline{PI} = 378\%$, which shows the potential cost savings through our approach.

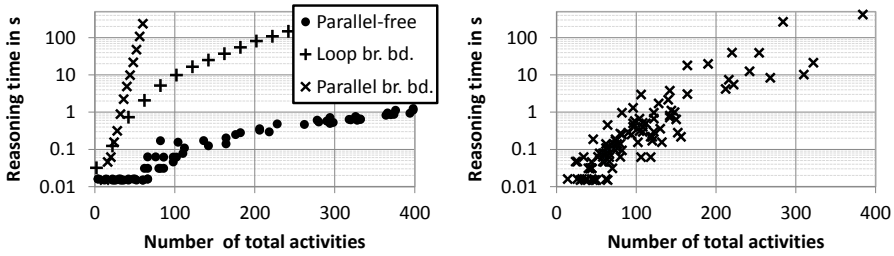

Fig. 3. Reasoning times on logarithmic scale

5 Related Work

The business process management community investigates the analysis of interaction properties [4], the difference analysis between process models [5], the recognition of equivalent fragments [6] and the validation of consistency-aware changes of a process model with respect to a process template [7]. However, none of these approaches considers the problem of process refinement given an abstract and a specific process.

Behavioural profiles [8] describe processes by characteristic relations. They are used for measuring the compliance of process executions, which are given by logs, with respect to their models [9]. Behavioural profiles offer an alternative representation compared to our predecessor and successor relationships. However, according to our notion of refinement, there are some particular cases like the occurrence of two exclusive activities in the same branch of a loop, which is differently handled in our loop break-down compared to the behavioural profiles.

A bunch of works [10–12] considers process model abstraction. A process model abstraction provides a more abstract and higher-level view by aggregating and eliminating activities of the original (more detailed) process model. In contrast to our work, several rules are used to preserve the execution order, while we allow for an arbitrary refinement and check the validity afterwards. Refinement of actions is modelled by operators in [13]. These operators preserve the semantic correctness by taking the relations into account. Our refinement notion differs in two aspects. First, we do not use refinement operators and therefore, we cannot ensure correctness by construction. Thus, a key part of our contribution is the validation of refinements. Second, our semantics rather refers to the interleaving semantics, where the behaviour is given by sequences of activities, while in [13] the causal semantics is used.

Work on process equivalence is faced with a related problem of formalising and comparing the behaviour of processes. A formalisation of equivalence for BPMN like process models is presented in [14]. The equivalence of process models, e.g., of a reference and a specific process, is analysed in [15], where equivalence is expressed by a degree of similarity between two processes.

Process algebra, rooted in transition and communication system modelling, serves as a formal specification of process behaviour in several works [16, 17]. Based on this formalisation, simulation and bisimulation allow the comparison of process behaviour regarding abstraction, specialisation and equivalence. Process decompositions, are limited to structured blocks. Thus, not all kinds of refinements in our work can be expressed by these formalisms.

6 Conclusion

In this paper, we have defined a formal semantics of process refinement based on the execution set semantics and presented an ontological solution to a process refinement problem and several reduction techniques to enable the refinement validation and explanation using standard reasoning services. The evaluation shows that our approach significantly improves the efficiency and correctness of process engineering.

Acknowledgement. This work was partially supported by the European Union's Seventh Framework Programme under grant agreement 604123 (FIspace).

References

1. Wyner, G.M., Lee, J.: Defining specialization for process models. In: Organizing Business Knowledge: The MIT Process Handbook, pp. 131–174. MIT Press (2003)
2. Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Instantaneous Soundness Checking of Industrial Business Process Models. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 278–293. Springer, Heidelberg (2009)
3. Curran, T.A., Ladd, T., Ladd, A.: SAP R/3 Business Blueprint: Understanding Enterprise Supply Chain Management, 2nd edn. Prentice Hall International (1999)
4. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing Interacting WS-BPEL Processes using Flexible Model Generation. DKE 64, 38–54 (2008)
5. Dijkman, R.: Diagnosing differences between business process models. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 261–277. Springer, Heidelberg (2008)
6. Gerth, C., Luckey, M., Küster, J.M., Engels, G.: Detection of Semantically Equivalent Fragments for Business Process Model Change Management. In: IEEE International Conference on Services Computing, pp. 57–64 (2010)
7. Sadiq, S., Orlowska, M., Sadiq, W.: Specification and Validation of Process Constraints for Flexible Workflows. Information Systems 30(5), 349–378 (2005)
8. Weidlich, M., Mendling, J., Weske, M.: Efficient Consistency Measurement Based on Behavioral Profiles of Process Models. IEEE Trans. Software Eng. 37(3), 410–429 (2011)
9. Weidlich, M., Polyvyanyy, A., Desai, N., Mendling, J., Weske, M.: Process Compliance Analysis based on Behavioural Profiles. Inf. Syst. 36(7), 1009–1025 (2011)
10. Smirnov, S., Reijers, H., Weske, M., Nugteren, T.: Business Process Model Abstraction: A Definition, Catalog, and Survey. Distributed and Parallel Databases, 1–37 (2012)

11. Eshuis, R., Grefen, P.: Constructing Customized Process Views. Data & Knowledge Engineering 64(2), 419–438 (2008)
12. Liu, D., Shen, M.: Workflow Modeling for Virtual Processes: An Order-preserving Process-view Approach. Information Systems 28(6), 505–532 (2003)
13. van Glabbeek, R., Goltz, U.: Refinement of Actions and Equivalence notions for Concurrent Systems. Acta Inf. 37(4/5), 229–327 (2001)
14. Lam, V.: Equivalences of BPMN Processes. Service Oriented Computing and Applications 3, 189–204 (2009)
15. van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.: Process Equivalence: Comparing Two Process Models Based on Observed Behavior. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 129–144. Springer, Heidelberg (2006)
16. Milner, R.: Communication and Concurrency. Prentice Hall (1989)
17. Sangiorgi, D.: Bisimulation for Higher-Order Process Calculi. Information and Computation 131, 141–178 (1996)