Author/s: Ruopeng Lu, Shazia Sadiq, Guido Governatori

Article title: On Managing Business Processes Variants

Journal: Data & Knowledge Engineering **ISSN:** 1872-6933

Year: 2009 **Volume:** 68 **Issue:** 7 **Pages:** 642-664

DOI to published version: https://doi.org/10.1016/j.datak.2009.02.009

On Managing Business Processes Variants ⋆

Ruopeng Lu [a], Shazia Sadiq [b], Guido Governatori [c]

[a] *SAP Research CEC Brisbane, Australia*

[b] *School of Information Technology and Electrical Engineering The University of Queensland, Brisbane, Australia*

[c] *Queensland Research Laboratory, National ICT Australia (NICTA) Brisbane, Australia*

Abstract

Variance in business process execution can be the result of several situations, such as disconnection between documented models and business operations, workarounds in spite of process execution engines, dynamic change and exception handling, flexible and ad-hoc requirements, and collaborative and/or knowledge intensive work. It is imperative that effective support for managing process variances be extended to organizations mature in their BPM (Business Process Management) uptake so that they can ensure organization wide consistency, promote reuse and capitalize on their BPM investments. This paper presents an approach for managing business processes that is conducive to dynamic change and the need for flexibility in execution. The approach is based on the notion of process constraints. It further provides a technique for effective utilization of the adaptations manifested in process variants. In particular, we will present a facility for discovery of preferred variants through effective search and retrieval based on the notion of process similarity, where multiple aspects of the process variants are compared according to specific query requirements. The advantage of this approach is the ability to provide a quantitative measure for the similarity between process variants, which further facilitates various BPM activities such as process reuse, analysis and discovery.

Key words: Business process management, process constraints, process variants, process similarity, flexible workflows

⋆ This research work has been conducted at The University of Queensland.

Email addresses: ruopeng.lu@sap.com (Ruopeng Lu), shazia@itee.uq.edu.au (Shazia Sadiq), guido.governatori@nicta.au (Guido Governatori).

1 Introduction

There have been many efforts towards providing agile business process management (BPM) support in recent years. Business process management systems (BPMS) have been recognized as a substantial extension to the legacy of workflow management systems (WFMS). While a typical WFMS supports process design, deployment and enactment, an extension of WFMS functionality provided by BPMS is the facilitation of process diagnosis activities [1]. Furthermore, new requirements emerging from the flexibility and dynamism of business processes require support for instance adaptation, which further impacts on the design, execution and especially the diagnostic activities of BPMS, and eventually will contribute to process evolution and improvement (cf. Fig. 1).

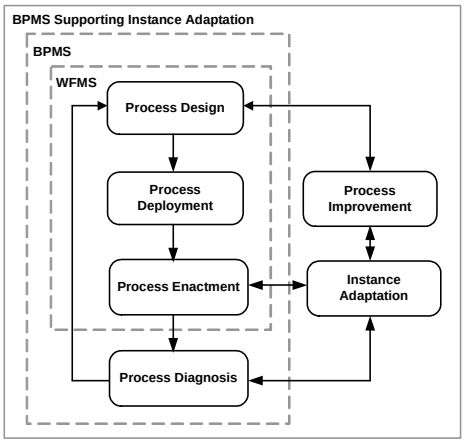


Fig. 1. The extended BPMS life-cycle supporting instance adaptation and process improvement

The process diagnosis phase refers to a wide range of BPM activities, including business process analysis (BPA) and process mining and discovery [1,2,10]. These post-execution activities are intended to identify and resolve operational process problems, discover preferred work practices, and provide business intelligence. Instance adaptation is an emerging paradigm due to various reasons such as changes in underlying business objectives and operational constraints, and unexpected events that cannot be handled by predefined exception handling policies. Consequently, the execution of process instances needs to be changed at runtime causing different instances of the same business process to be handled differently.

Over the last several years of developments in BPM research and industry, we see two equally strong but often conflicting forces impacting on the developments. Where as one fundamental aspect of BPMS and its predecessor WFMS, is to provide control and coordination of business activities, there is another equally demanding aspect of ensuring that the control does not prohibit the

operational flexibility, to unacceptable levels.

There are many use cases for such requirements. For example, in in-patient hospital administration processes, where patient admission procedures are predictable and repetitive, however, in-patient treatments such as x-rays, pathology tests etc. are prescribed uniquely for each case, but none-the-less have to be coordinated and controlled. Another example can be found in higher education and professional training, where students with diverse learning needs and styles are working towards a common goal (degree program). Study paths taken by each student need to remain flexible to a large extent, time providing study guidelines and enforcing course level constraints is necessary to remain compliant with curriculum requirements and maintain a certain quality of learning.

Similarly, consider an engineering firm that provides maintenance and advisory services for telecommunication faults and inquiries. Service plans for individual customer inquiries will be quite diverse, even though basic administration may be the same. We will later introduce the last scenario in more detail as it is used as a running example to demonstrate various concepts and methods.

Many research prototypes (MOBILE [18], ADAPT$_{flex}$ [31], Pocket of Flexibility [35], Worklets [5], DECLARE [30]) have shown a variety of conceptually advanced solutions along this direction (see Section 6 for detail descriptions). In order to provide a balance between the opposing forces of control and flexibility, we have argued for [35], a modeling framework that allows part of the model that requires less or no flexibility for execution to be predefined, and part to contain loosely coupled process activities that warrant a high level of customization. When an instance of such a process is created, the process model is *concretized* by the domain expert at runtime. The loosely-coupled activities are given an execution plan according to instance-specific conditions, possibly some invariant process constraints, and their expertise. Current BPM solutions only provide limited support for instance adaptation. For example, the de facto industrial standard for process modeling, Business Process Modeling Notations (BPMN) [29] offers a concept called Ad Hoc Sub-process (AHS) that provides certain level of support for instance adaptation requirements. AHS is a group of activities that have no pre-defined execution dependencies. A set of activities can be defined for the AHS, but the sequence and number of executions for the activities is completely determined by the performers of the activities and cannot be defined at design time. Based on the runtime conditions and their domain expertise, the performers determine how to execute the activities within the AHS, namely the order of execution (sequential or parallel). The contained activities can be executed multiple times until the pre-defined completion conditions are satisfied [29]. Fig. 2 shows an example process model with a AHS for a network diagnostics scenario. Fig. 3 shows

three of many potential execution possibilities of the AHS [1] .

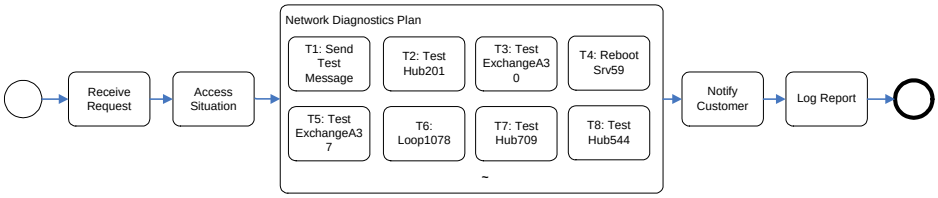
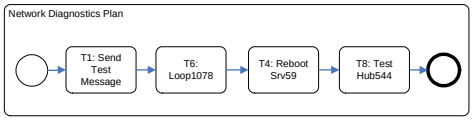
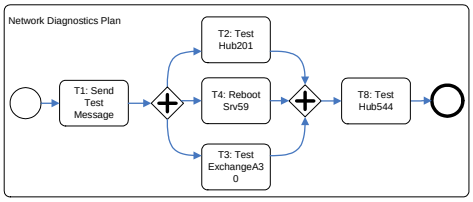


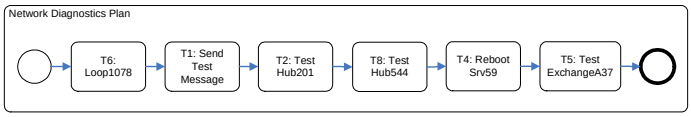
Fig. 2. A network diagnostics scenario modeled with a BPMN ad hoc sub-process



(a) An execution scenario where four tests are chosen to execute in sequence



(b) An execution scenario where three tests are chosen to execute in parallel after sending test message, and concluding by testing Hub544



(c) An execution scenario where six tests are chosen to execute in sequence

Fig. 3. Execution possibilities of the BPMN ad hoc sub-process shown in Fig. 2

The subprocess illustrated in these figures can be considered part of a process that manages maintenance and advisory services for telecommunication faults and inquiries in a telecommunications company. Below we detail a typical scenario in this regard to provide further motivation and rationale for the approaches proposed in this paper. This scenario will constitute a running example throughout the paper:

A telecommunications company receives customer enquiries about network connection problems, where each complaint case is assigned to a system engineer who is responsible for designing a service plan and solving the problem. The inquiry logging and reporting procedures are predictable and repetitive, while diagnostic tests required to prepare a service plan will typically be case specific and potentially uniquely configured for each case, but nonetheless still have to be coordinated and controlled. The particular configuration of diagnostic tests specific to a given instance is expected to be determined dynamically by a domain expert, such as a senior engineer, based on case specific properties and the experts knowledge and experience.

[1] Note the modeling notation is BPMN [29].

There are eight diagnostic tests $T1, T2, \ldots, T8$ available in the process, as shown in Fig. 2:

- Send Test Message ($T1$)
- Test Hub201 ($T2$)
- Test ExchangeA30 ($T3$)
- Reboot Srv59 ($T4$)
- Test ExchangeA37 ($T5$)
- Loop1078 ($T6$)
- Test Hub709 ($T7$)
- Test Hub544 ($T8$)

Any number of these tests can be prescribed for a given request, in some preferred order. The network engineer has the flexibility to design a service plan that best suits the individual case. The design decisions can only be made at runtime when case specific conditions are available and thus cannot be fully anticipated at design time.

Although constructs such as BPMN AHS can be flexibly configured to execute contained activities sequentially or in parallel, there is no means for controls such as restricting the number of selectable activities, nor defining complex/partial dependencies among them. Techniques are required where part of the modeling effort is transferred to domain experts or knowledge workers who make design decisions at runtime under meaningful, domain-relevant constraints. In Section 3, we will provide an approach capable of capturing a large number of selection and scheduling constraints, thus providing a meaningful context for runtime instance adaptations.

At the same time, it can be observed that the typical consequence of an effective instance adaptation environment is the production of a large number of process variants. An executed process instance reflects a variant of realization of the process constraints, and provides valuable knowledge of work organization at the operational level. There is evidence that work practices at the operational level are often diverse, incorporating the creativity and individualism of knowledge workers and potentially contributing to the organization's competitive advantage. Such resources can provide valuable insight into work practice, help externalize previously tacit knowledge, and provide valuable feedback on subsequent process design, improvement, and evolution.

Nevertheless, the way that domain experts reason about the situation during instance adaptation cannot be truly reconstructed using computational techniques. Building a repository to systematically capture, structure and subsequently deliberate on the decisions that led to a particular design is a more pragmatic way to approach the problem. We observe that a process variant at least contains information from the following dimensions:

- **Structural** dimension contains the process model based on which the process instance is executed. For process variants, the structural dimension is represented by the process model that is adapted from the design time model for the particular process variant during instance adaptation.
- **Behavioral** dimension contains executional information such as the set of tasks involved in the process execution (may differ from structural dimension due to choice constructs), the exact sequence of task execution, the performers and their roles in executing these tasks, the process-relevant data, and execution duration of the process instance.
- **Contextual** dimension contains descriptive information (annotations) from the process modeler about the reasoning behind the design of a particular process variant.

There are various occasions in the BPM life-cycle when precedents of process variants need to be retrieved. For example, during instance adaptation itself, domain experts may refer to a list of precedent process variants designed for a similar situation. Using appropriate analysis techniques, a collection of sufficiently similar process variants could be generalized as the preferred/successful work practice, and consequently contribute to the design of a given instance and subsequently to process improvements.

In this paper we will address both the above issues, namely techniques for supporting *instance adaptation*, and the utilisation of the direct consequence of instance adaptation - *management of process variants*. We will introduce a framework for instance adaptation to support flexible business process management based on the notion of *process constraints*. This approach transfers part of the process modeling effort to domain experts who make execution decisions at runtime. Instance adaptation is supported by techniques for specifying instance-specific process models and constraint checking in different variants of the business process. We will demonstrate how the specification of so-called selection and scheduling constraints can lead to increased flexibility in process execution, while maintaining a desired level of control. This aspect is detailed in Section 3.

We then introduce the key technique for managing process variants, namely a query formalization and progressive refinement technique for *process variant retrieval*. In our previous work, we have developed a reference architecture for managing such process variants for effective retrieval [23]. The contribution of this paper is to provide an approach for utilizing the retained process variants. An essential concept in this regard is the definition of *similarity* between process variants in terms of their various dimensions. In other words, how to characterize the degree of match between two similar process variants. This is a hard problem in general due to the informal nature of commonly adopted process description languages, and more so due to the subjectivity in process model conceptualization. Questions such as how to measure the similarity be-

tween two process variants having different process models but same sequence of task execution can come forth. From the behavioral perspective two variants are equivalent since they have the same execution behavior. While from the structural perspective they may be dissimilar. Thus variants can share features in one dimension but be dissimilar in another dimension, making an objective evaluation of similarity rather difficult. At the same time, it is desirable that the similarity between the variants can be quantified, i.e., to be able to define a metric space to indicate the degree of similarity or dissimilarity.

The rest of the paper is organized as follows. Section 2 will provide background concepts for the underlying framework for supporting instance adaptation. In Section 3, we introduce the core concept of process constraints, based on which the instance adaptation framework is developed. Section 4 discusses how a repository of process variants manages a large number of executed process variants as an information resource. In particular, the schema for process variants in the repository will be defined. Queries applicable on process variants and their formalization will be discussed in Section 5. In this section, we also provide a quantitative measure for defining similarity between process variants, covering structural, behavioral and contextual dimensions, as well as a progressive-refinement technique for query processing. Related work is presented in Section 6, followed by the conclusion and future work in Section 7.

2 Framework for Managing Business Process Variants

The framework for constraint-based flexible business process management comprises of two major components, namely Business Process Constraint Network (BPCN), and Process Variant Repository (PVR). In this section, the overall approach of the framework is presented, which shows how this framework can be applied to BPMS, and the relationship between BPCN and PVR. The functionality of the proposed framework (cf. Fig. 4) is explained with respect to different stages in the BPM life-cycle.

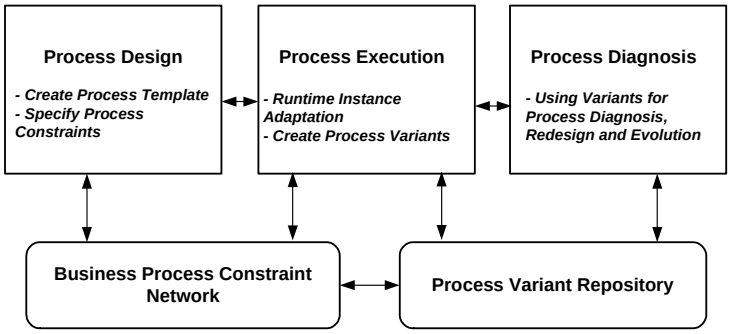


Fig. 4. Framework for constraint-based business process management

The rationale of BPCN is to provide a descriptive way to build models for business processes where the complete process cannot be prescribed at design time. Essentially, BPCN captures the set of available tasks in a given business process, and allows for a specification of constraints to govern how instance-specific process models can be built from these tasks. This approach relaxes rigid process descriptions into a set of minimal constraints, such that a variety of process models can be designed for different process instances, provided the relevant constraints are satisfied. The basic intuition and working of BPCN is described below.

The flexible business process is defined at *design time*. The process modeler designs a *template* for the underlying business process, which contains a pool of available tasks in the business process, and the specification of *process constraints* restricting how these tasks can be selected and executed. This may be part of a larger process model defined in a conventional or prescriptive way (see e.g. the example in Fig. 5), or it may constitute the entire process model. In the absence of a process model, the specification of process constraints reflects the process logic and hence the template and model become synonymous. However, the constraint representation of the process logic has a descriptive nature, which specifies *what* must be done but not *how*. The quality of the process constraints is then checked by constraint reasoning techniques, such that the validity of the constraints is ensured (Section 3). The definition created above is uploaded to the process engine. This process is now ready for execution.

At *runtime*, the user or application would create an instance of the defined process. On instantiation, the engine creates a copy of the process definition and stores it as an instance template (instance-specific process model). This process instance is now ready for execution. The available process activities of the newly created instance are assigned to performers through work lists and activity execution takes place as usual, until the instance needs to be dynamically adapted to particular requirements arising at runtime.

Instance adaptation takes place when a domain expert wishes to create an instance-specific variant of the process. The domain expert undertakes the task of dynamically adapting the instance template with available pool of activities, while guided by the specified constraint set. This revises the instance-specific process model and requires a build function which has the capability to load and revise instance templates for active instances. The instance-specific process model is referred to as a *process variant*.

The next step is to verify the new model representing the process variant, to ensure that it conforms to the correctness properties of the modeling language as well as the given constraints. For the adapted process variant, conformance to constraints is checked through the verification component of BPCN (see Sec-

tion 3.4). On satisfactory verification the newly defined (or revised) instance resumes execution. Execution will now continue as normal, until completion or until re-invocation of the build function, in which case instance adaptation will be performed again.

BPCN is essentially a design approach, but is supported by an execution environment in which process instances can be individually specified according to specific needs, but still conform to process constraints, e.g., a particular configuration of tests prescribed by a service plan. The execution environment allows for the generation of potentially a large number of customized process variants, each of which has been constructed with the help of a domain expert utilizing expert knowledge as well as case-specific requirements.

After the process variant has been executed, the adapted process model and its runtime properties are acquired by PVR, which is a structured repository of storing executed process variants. PVR also provides support for reusing the retained information. Thus, PVR provides the support for process diagnosis and subsequent process improvement activities. These functions of the PVR will be discussed in Sections 4 - 5.

3 Business Process Constraint Network (BPCN)

The foremost factor in designing business processes is achieving improvements in the business outcomes [17]. However, decisions at the strategic level need to be evaluated in light of constraints that arise from several sources. It has been identified that at least four sources of constraints have impact on a business process design:

- **Strategic constraints** define the tactical elements of the process e.g. approval of director required for invoices beyond a certain value.
- **Operational constraints** are determined through physical limitations of business operations, e.g. minimum time for warehouse offloading.
- **Regulatory constraints** are prescribed by external bodies and warrant compliance e.g. financial and accounting practices (Sarbanes-Oxley Act), or batch identification for FDA in the pharmaceutical industry.
- **Contractual constraints** define the contractual obligations of an organization towards its business partners, e.g. maximum response time for a service.

In order to harness the full power of BPM techniques, each of these constraints should eventually be translated into constructs of a (executable) business process model, and subsequently lead to process enforcement at the business activity level. In BPCN, constraints can be modelled according to specific

business requirements in some business domain by a process modeller, and a constraint network is then formulated according to the properties of the related business domain. This can be referred to as *constraint modeling*. In this approach, the process modellers manually transform the process knowledge into the executable form (as BPCN constraints). However, in the application areas where business process requirements are represented in a well-defined format, such as business contract or regulatory obligations, it is desirable for a systematic methodology for transforming well-defined contractual expressions to a set of executable constraints. This transformation process is referred to as *constraint acquisition*.

BPCN primarily provides the environment for constraint modeling in order to facilitate the instance adaptation framework introduced in the previous section. In BPCN, business process specification is undertaken through the specification of process constraints. Under the same process template, the selection, and the subsequent modeling of selected tasks can be largely different from instance to instance. When adapting a process variant at runtime, the first step is to choose a suitable set of tasks to execute in the current process instance. There are some core tasks that need to be executed in every instance, while some others may be optional and can be included/excluded for execution according to user preference and instance-specific conditions. At the same time, there are many other restrictions and inter-dependencies that need to be expressed. These are referred to as *Selection* constraints, and essentially define what activities constitute the process model. In this paper, the focus of BPCN is on a range of elementary controls at task level, i.e., the individual property and the binary relations among the tasks in a process template. Accordingly, 9 classes of selection constraints have been conceptualized for expressing such restrictions on task selection.

Furthermore, there are constraints that define how the selected activities are performed, both in terms of ordering as well as temporal dependencies. These are referred to as *Scheduling* constraints. These constraints are applicable at process level, which constitute the specification of task selection restrictions and the dependencies within these tasks including control dependencies (such as sequence, alternative, parallel etc.) and inter-task temporal dependencies (such as relative deadlines). Such constraints are defined in graphical notations provided by BPCN, which can be used to conceptually express requirements for instance adaptation by general process modelers who have little or no formal background. With the help of examples, this section briefly introduces how to apply business process constraints to achieve flexibility in process management.

Consider the process given in the figure. This represents the scenario introduced in Section 1. Recall that any number of these tests can be prescribed for a given request, in some preferred order. The network engineer has the

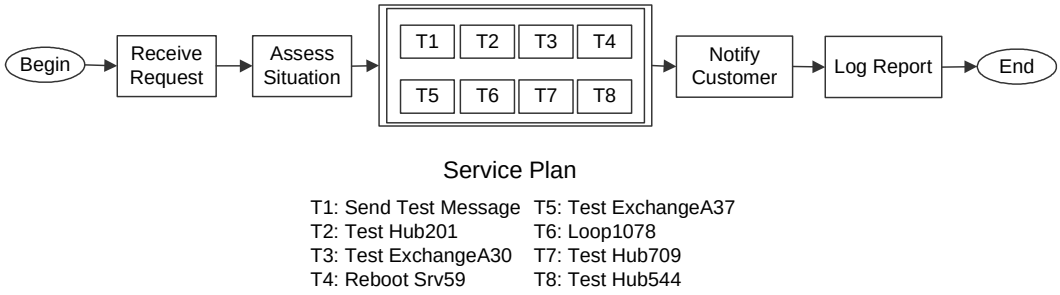


Fig. 5. Process model containing a process template for the network diagnosis process

flexibility to design a service plan that best suits the individual case. The design decisions can only be made at runtime when case specific conditions are available and thus cannot be fully anticipated at design time. At the same time, some dependencies and restrictions among the available tests can be abstracted beforehand, such as below:

- $T1$, $T4$ and $T8$ are basic tests and must be performed for all testing cases;
- Based on operational guideline, minimal 4 tests have to be performed to guarantee accuracy, while maximal 6 tests can be selected for efficiency;
- Testing ExchangeA30 ($T3$) depends on the result of test data of Hub201 ($T2$), and must be performed after Hub201;
- Either Hub709 ($T7$) or Hub544 ($T8$) needs to be tested as they are located on the same communication bus;
- ExchangeA30 can only be tested ($T3$) when Srv59 is rebooting ($T4$).

The abovementioned restrictions and dependencies can be considered as constraints on tests selection (i.e., the requirements on mandatory selection, and the number of tests) and scheduling (the order of execution). However, it is obvious that even with a small number of activities, there is a large number of possible service plans that satisfy the (quite restrictive) testing constraints. For example, while at least four and at most six tests can be chosen out of eight available ones, where there are three mandatory tests that have to be chosen all the time, the engineer can still choose to perform exactly four, five or up to six tests. In combinatorial terms, there are $C_1^5 + C_2^5 + C_3^5 = \frac{5!}{1!(5-1)!} + \frac{5!}{2!(5-2)!} + \frac{5!}{3!(5-3)!} = 5 + 10 + 10 = 25$ possible selection scenarios. Reducing the number from scenarios where $T3$ is selected without $T2$, or $T7$ and $T8$ are selected together (14 scenarios), there are still 11 possible selection scenarios (e.g., choosing tests $T1$, $T4$, $T8$ and $T5$; or $T1$, $T4$, $T8$, $T2$, $T3$ and $T6$). Consider that for each such scenario, the engineer can further define a variety of execution plans including any execution order, e.g., execute all selected tasks in sequence, or in parallel, or some in sequence and others or parallel. In graphical modeling notation (e.g., BPMN), it is tedious if not impossible to capture and manage all such scenarios and to articulate the conditions for the alternative branches.

This is a typical example of the need for operational flexibility. In our approach, it is assumed that the underlying system supports a typical graph-based process model and a state-based execution model. Such process models support typical constructs such as sequence, fork, choice, etc., and activity execution is based on a finite state machine with typical states such as available, commenced, suspended, and completed. This is a common environment for many commercial process management systems, and therefore our approach has minimal impact on underlying systems. For the subsequent discussion on process constraints, we first introduce some basic terms:

A process template PT is defined by a set of process tasks T and a set of process constraints C. T represents the available pool of activities to be adapted at runtime. C is a set of (selection and/or scheduling) constraints that defines relations between the properties of tasks in T.

3.1 Selection Constraints

We now define various selection constraints and their graphical notations that can be used to conceptually express task selection requirements. The notations are developed for process modelers who have little or no background in constraint satisfaction theory. Nevertheless, formalization of selection constraints is later presented, in order to precisely define the semantics.

The following classes of selection constraints have been identified:

Mandatory constraint *man* defines a set of tasks that *must be executed* in every process variant, in order to guarantee that intended process goals will be met.

Prohibitive constraint *pro* defines a set of tasks that *should not be executed* in any process variant.

Cardinality constraint specifies the *minimal min* and *maximal max* cardinality for selection among the set of available tasks.

Inclusion constraint *inc* expresses the *dependency* between two tasks T_x and T_y, such that the presence of T_x imposes restriction that T_y must also be included. **Prerequisite** *pre* constraint is the inverse of an inclusion constraint.

Exclusion constraint *exc* prohibits T_y from being included in the process variant when the T_x is selected.

Substitution constraint *sub* defines that if T_x is not selected, then T_y must be selected to *compensate the absence* of the former.

Corequisite constraint *cor* expresses a stronger restriction in that either both T_x and T_y are selected, or none of them can be selected, i.e., it is not possible to select one task without the other.

Exclusive-Choice constraint *xco* is also a more restrictive constraint on the selection of alternative tasks, which requires at most one task can be selected from a pair of tasks (T_x, T_y).

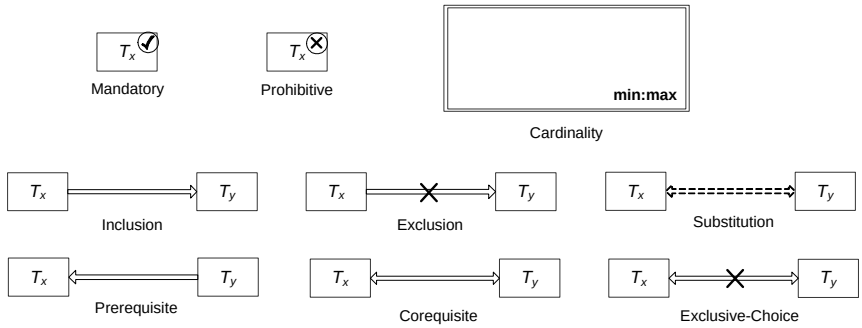


Fig. 6. Notations for selection constraints

The notation for selection constraints is summarized in Fig. 6. Note however, the arrows do not indicate order of execution for the tasks within the process template (i.e., control flow structure of the tasks). Rather, they are introduced to provide visual representation for selection constraints, as indicated by the labels.

Below we provide the formal consideration for selection constraints. Let $T = \{T_1, T_2, \ldots, T_n\}$ denote the set of all tasks in a process template PT. Each task T_i is considered as a propositional variable ranging over domain $D_i = \{0, 1\}$. Let $T_i = 1$ stand for the *presence* of task T_i in a process variant V and $T_i = 0$ stand for *absence*.

Mandatory, prohibitive and cardinality constraints can be defined by restricting the domains of respective tasks. A mandatory task $T_i \in T$ is denoted by $man(T_i)$, where ***man*** is a property of T_i restricting its domain $D_i = \{1\}$. The set of all mandatory tasks in a process template PT is given by:

$$R^{man} = \{T_i \mid man(T_i)\}$$

A task T_i in a process template PT is prohibitive if it is forbidden to be selected in any process variants V of PT. A single prohibitive task T_x can be denoted by $pro(T_x)$, where pro is a property of T_x restricting $D_x = \{0\}$. The set of all prohibited tasks in a process template PT is given by:

$$R^{pro} = \{T_i \mid pro(T_i)\}$$

A *minselect* constraint is denoted by $R^{min(m)} \subseteq T$, such that $|R^{min(m)}| \geq m$, and $\forall T_i \in R^{min(m)}, D_i = \{1\}$. The *minselect* constraint restricts that every process variant V should contain all tasks in R^{man}, and zero, one or more

tasks from $(T - R^{man})$. A *maxselect* constraint is denoted by $R^{max(n)} \subseteq T$, such that $|R^{max(n)}| \leq n$, and $\forall T_i \in R^{max(n)}, D_i = \{1\}$.

The mandatory, prohibitive and cardinality constraints are defined by restricting the domain of a single task. On the other hand, the other selection constraints including inclusion, exclusion, substitution, prerequisite, corequisite and exclusive-choice are binary relations that are defined by restricting the domains of the pair of tasks. We accordingly call them containment constraints. For example, An inclusion constraint R^{inc} is a binary relation on a pair of variables (tasks) $T_x, T_y \in T$, if and only if (*iff*):

$$R^{inc} = ((T_x, T_y), \{(0,0), (0,1), (1,1)\})$$

An inclusion constraint R^{inc} defined on tasks T_x, T_y reads T_x includes T_y. By definition, it restricts the domain of values that can be assigned to the pair (T_x, T_y). In this case, either $(0,0)$, $(0,1)$, or $(1,1)$ can be assigned. Applying this definition to task selection, it expresses that when T_x is selected, T_y must also be selected (T_x is the dependent of T_y). The following selection scenarios are permitted:

- neither T_x nor T_y is selected, i.e., (0, 0);
- T_y is selected without T_x, i.e., (0, 1);
- both T_x and T_y are selected, i.e., (1, 1).

The scenario $(1, 0)$ is prohibited where T_x is selected without T_y, thus enforcing the inclusion relationship between selection of T_x and T_y.

Similarly, an exclusion constraint R^{exc} is a binary relation on a pair of variables $T_x, T_y \in T$, *iff*:

$$R^{exc} = ((T_x, T_y), \{(0,0), (0,1), (1,0)\})$$

An exclusion constraint prohibits the selection scenario $(1, 1)$ where both T_x and T_y are selected. Table 1 presents a summary for the definition of the containment constraints.

3.2 Scheduling Constraints

Scheduling constraints take advantage of the temporal property of each task in a process template, namely the *execution duration*. Suppose each task T_i is given an expected execution duration $dur(T_i)$. It can be observed that representing each task by its execution duration (as an interval), the interdependencies between these tasks can be modeled [6].

Table 1
Definitions of Containment Constraints

Constraint	Definition
R^{inc}	$((T_x, T_y), \{(0,0), (0,1), (1,1)\})$
R^{exc}	$((T_x, T_y), \{(0,0), (0,1), (1,0)\})$
R^{sub}	$((T_x, T_y), \{(0,1), (1,0), (1,1)\})$
R^{pre}	$((T_x, T_y), \{(0,0), (1,0), (1,1)\})$
R^{cor}	$((T_x, T_y), \{(0,0), (1,1)\})$
R^{xco}	$((T_x, T_y), \{(0,1), (1,0)\})$

The following classes of scheduling constraints have been identified:

Before constraint expresses that tasks T_x and T_y should be arranged sequentially, but not in adjacency in the process model.

Meet constraint expresses that T_x and T_y are to be arranged sequentially, and in adjacency, i.e., T_x and T_y need to be placed in adjacent to each other (consecutive placement).

Order constraint is less restrictive that expresses T_x and T_y are to be executed in a specific order (not necessarily in adjacency). If a pair of tasks (T_x, T_y) satisfies either *Before* or *Meet*, it also satisfies *Order* constraint.

Starts constraint expresses a restriction on commencement time of executing tasks T_x and T_y, such that T_x and T_y are arranged in parallel, and commence execution at the same time. In addition, T_x should finish execution before T_y.

Finishes constraint expresses a restriction that T_x and T_y are arranged in parallel, and complete execution at the same time. In addition to both tasks completing execution at the same time, T_x should commence execution after T_y.

During constraint expresses that T_x and T_y are arranged in parallel, and T_x can only commence execution after T_y has commenced, and must complete execution before T_y completes.

Equals constraint restricts that T_x and T_y are arranged in parallel, and commence and complete execution at the same time.

Parallel constraint is also less restrictive that expresses T_x and T_y are arranged in parallel. If a pair of tasks (T_x, T_y) satisfies either *Starts*, *Finishes*, *During*, or *Equals* constraint, it also satisfies *Parallel* constraint.

The set of scheduling constraint notations is summarized in Fig. 7.

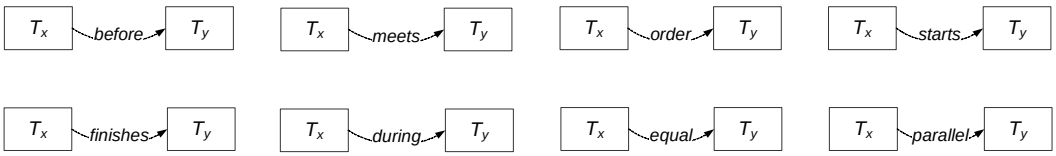


Fig. 7. Notations for scheduling constraints

3.3 Validation of Constraints

We now use the running example introduced previously to demonstrate constraint specification and validation. Suppose a set of selection and scheduling constraints (cf. Fig. 8) is specified for the process template in Fig. 5 containing following constraints.

- *Mandatory* task $T1$, $T4$ and $T8$;
- *Minimal* selection of 4 and *maximal* selection of 6 tasks;
- $T2$ is the *prerequisite* of $T3$;
- $T7$ and $T8$ are of *exclusive-choice*;
- *Order* between $T2$ and $T3$;
- $T3$ *starts* $T4$.

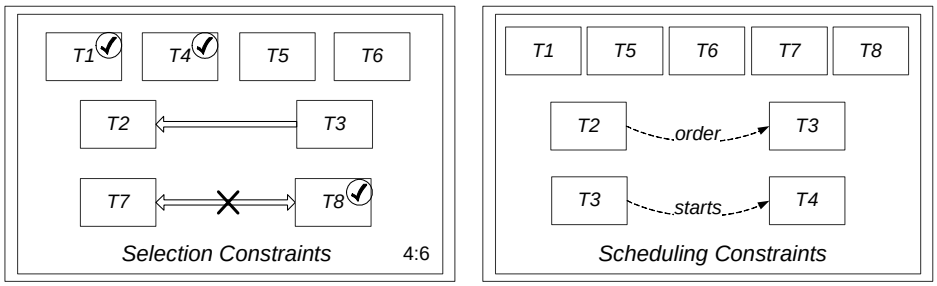


Fig. 8. Selection and scheduling constraints defined on the pool of available tasks

Given the constraint specification in a process template, it is necessary to validate the constraint set, or ensure that the constraints are satisfiable, i.e., there exists at least one solution for the constraint set. At the very least, the validation of the constraint set must firstly ensure that any hidden facts in the constraint specifications are explicitly shown. This makes the constraint specification non-redundant and understandable, and is particularly useful for domain experts when concretizing the process template during instance adaptation. The second requirement is to ensure that there is at least one possible scenario where no constraint is violated. In other words, to avoid contradicting constraint specifications. For example, T_x *includes* T_y and T_x *excludes* T_y are two contradicting constraints. If such conflicts exist, it is not possible to find a scenario where no constraint is violated. Furthermore, it is possible that the selection constraint specification is too restrictive that there is no possible way for satisfactory task selection. An obvious example can be a process template with prohibitive constraints on every task. In practice however, it is expected

that a large number of instance adaptation scenarios will be typically satisfied from a given set of constraints.

In order to provide a means of validating the constraints, the constraint specification is transformed into respective constraint systems (namely SCN and TCN for selection and scheduling constraints). BPCN uses Constraint Satisfaction [11] as the underlying theory to formally model and reason about process constraints. Selection (SCN) and scheduling constraint networks (TCN) are defined to provide formal semantics to validate the constraint specifications. SCN and TCN can also be used to check for the conformance of the adapted process variants at runtime. This is ensured through the enforcement of network consistency in SCN and TCN. SCN is a binary Boolean constraint network, and TCN is a qualitative temporal constraint network. The complexity of the consistency checking algorithm is bound by the number of constraints in the network. Formal specifications of process constraints can be found in [24], where definitions of SCN, TCN, as well as the constraint validation and process variant verification techniques are discussed in detail.

Below we present the basic intuition behind the validation procedure using the running example, where in there are some implicit (hidden) and redundant constraints. An exclusive-choice constraint is defined on $T7$ and $T8$, which means either $T7$ or $T8$ can be included but not both. At the same time, $T8$ must be chosen for all process variants (mandatory task). The implicit constraint is $T7$ to be prohibitive, which also makes the exclusive-choice constraint redundant. Furthermore, an implicit constraint $T2$ *order* $T4$ can be implied from $T2$ *order* $T3$ and $T3$ *starts* $T4$. This is because when $T3$ and $T4$ start execution at the same time, if $T2$ finishes execution on or before $T3$ starts, the same relation also applies to $T2$ and $T4$ (transitivity). The validated constraint specification is shown in Fig. 9, where the redundant constraint is removed, and implicit constraints are made explicit.

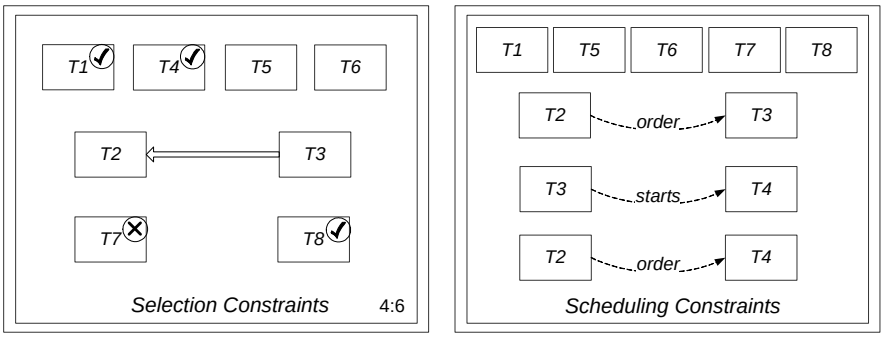


Fig. 9. Validated selection and scheduling constraints defined in Fig. 8

The process constraints in BPCN express minimal restrictions on task selection and scheduling for all business process variants. With the graphical notations, the constraint specification is used to provide visual support for domain experts to design process variants during instance adaptation. This section demonstrates how instance adaptation is supported at runtime.

Instance adaptation is generally performed in two steps, namely task selection and task scheduling. A domain expert first chooses from the task pool, the set of tasks to be performed in the current process variant. Based on which, the domain expert can then assign an expected execution duration for each task and then define the flow dependencies for the selected tasks.

As a result, each process variant contains a complete process model (cf. Definition 1 in Section 4). A process variant is executable in that the process model with flexible components have been concretized, i.e. the set of tasks to be executed, and the assignment of exactly one temporal relation between each pair of tasks are defined in the process model. Any user specified process variant needs to be checked against the given process constraints, i.e., a process variant is consistent if the task selection and schedule satisfy the constraint specification.

Fig. 10 shows four concretized process variants for the example process template (cf. Fig. 5). For ease of discussion, the execution durations are annotated with each task in the figure. These variants are to be verified against the selection (displayed on lower left) and scheduling constraints (lower right). The goal of verification is to first check whether the set of selected tasks satisfies relevant selection constraints, and secondly whether the execution schedule satisfies relevant scheduling constraints.

In Fig. 10, variants V_2, V_3 and V_4 satisfy all relevant selection and scheduling constraints. In particular, variant V_2 contains the minimal number of (4) tasks required, which includes all mandatory tasks $T1$, $T4$ and $T8$. Since task $T2$ is not included, the only applicable scheduling constraint $T3$ *starts* $T4$ is satisfied. Variants V_3 and V_4 contain the maximal number of (6) tasks which also include all mandatory tasks. All three scheduling constraints are applicable and satisfied.

On the other hand, variant V_1 violates the prohibitive constraint since it contains $T7$. Assumed that there is no transition overhead between tasks in the process model. V_1 violates the order constraints between $T2$, $T3$, and $T2$, $T4$. This is because $T2$ cannot start before $T7$ finishes (it takes 20 *mins* after $T1$ finishes to start $T7$), while $T3$ and $T4$ can start execution before $T2$ (it takes 10 *mins* after $T1$ finishes to start $T3$ and $T4$). To resolve this violation, $T7$

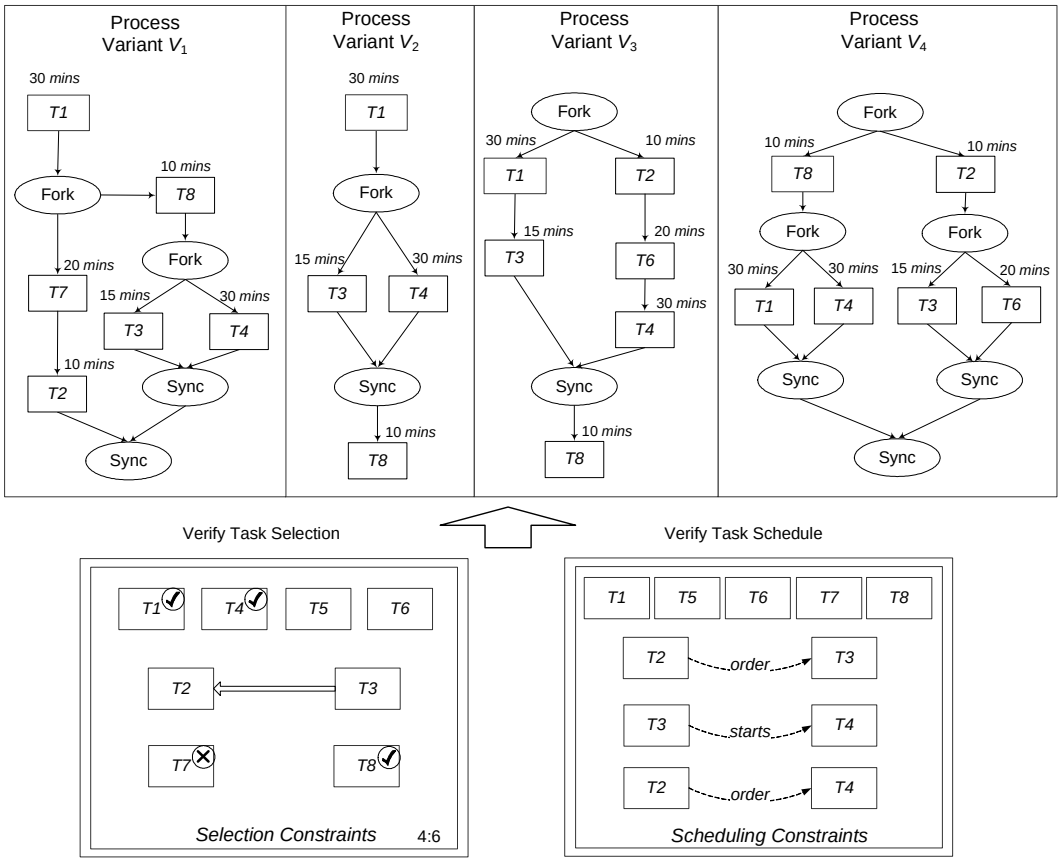


Fig. 10. Verifying process variants against process constraints

can be removed from the model, and the previously violated constraints can be satisfied. The complexity of the verification algorithms is bound by the number of constraints in the constraint network.

3.5 Summary

This section has presented how selection and scheduling constraints can be specified at *design time*, through intuitive constraint notations. The quality of the constraint specification is checked through the formal machinery of SCN and TCN respectively. This section further demonstrates the applicability of BPCN, which aims at presenting how instance adaptation is supported at *runtime*. Different process models can be built/tailored for individual process instances at runtime. Such instance-specific process models (process variants V) contain a complete process model that is defined by domain experts at runtime during instance adaptation. With the possibility for generating a large number of such process variants from the execution environment, the following section on process variant repository will demonstrate how such valuable information can be managed and reused.

4 Process Variant Repository

It can be observed that BPCN facilitates the creation of "quality" process variants, since each variant conforms to a set of necessary constraints, but also represents a domain expert's preferred approach to handle a particular case. In the proposed framework, process variants and their properties will be retained in a repository, called Process Variant Repository (PVR). Over time, PVR can build into an immense corporate resource.

The fundamental goal of PVR is to provide an appropriate characterization to describe the preferred work practices represented through process variants, and subsequently generalize the conditions contributing to the preference. PVR provides a well-formed structure to retain past process designs, as well as an instrument to utilize the adaptations manifested in process variants. In particular, a facility for discovery of preferred variants is provided through effective search and retrieval based on the notion of process similarity, where multiple aspects of the process variants are compared according to specific query requirements.

4.1 Reference Architecture

The capture of executed process variants and the subsequent retrieval of preferred process variants are the two major functions of PVR [23]. Fig. 11 presents an overview of the PVR reference architecture.

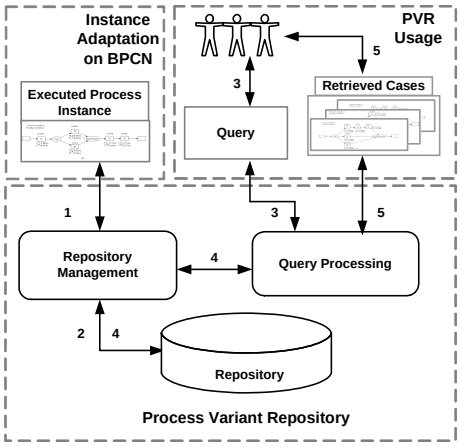


Fig. 11. Reference architecture of PVR

In Step 1-2 as annotated in Fig. 11, an executed process variant is retained in the repository. The variant is created using the instance adaptation mechanism of BPCN. Later, a query is formulated to specify variant retrieval requirements (Step 3). The query requirement is formulated with the help of the query

processing component. In Step 4, the repository is searched to find matching process variants according to query requirements. The goal of this step is to retrieve a set of sufficiently similar process variants to the query. In Step 5, the best matches are selected from the set of initial matches according to the degree of similarity relevant to the query. The further selection involves a ranking process. Step 4-5 will be repeated if a progressive refinement approach is taken, where the initial query definition is refined in order to obtain a more restrictive set of results.

4.2 Process Variant Schema

In order to provide a query facility for variants, it is necessary to first define a schema for describing them. The particular design of a variant is reflective of a domain expert's preferred work practice. However, (given groups of) variants are derived from a common design time process templates and hence can have a significant overlap as well. Before we present the schema of process variant, we first define two important concepts, including process model and execution sequence.

Definition 1 (Process Model) *A process model W is a pair (N, F), which is defined through a directed graph consisting a finite set of nodes N, and a finite set of flow relations $F \subseteq N \times N$. Nodes are classified into tasks T and coordinators C, where $N = C \cup T$, and $C \cap T = \emptyset$. T is the set of atomic tasks in W, and C contains coordinators of the type: $Begin$, End, $Fork$, Syn, $Choice$, $Merge$, where*

- *$Begin$ node represents the beginning of the process model;*
- *End node represents the end of the process model;*
- *$Choice$ node represents the divergence of a single path into two or more mutually exclusive alternative paths (cf. XOR-Split, Exclusive Choice);*
- *$Merge$ node represents the convergence of two or more mutually exclusive alternative paths into a single path (cf. XOR-Join, Simple Merge);*
- *$Fork$ node represents the divergence of a single path into two or more parallel paths (cf. AND-Split, Parallel split);*
- *Sync node represents the convergence of two or more parallel paths into a single path (cf. AND-Join, Synchronization).*

A *sub-process* (or a *process component*) is a special type of W, which is a fragment of a process model in which *Begin* and *End* are excluded from its coordinator nodes. Task nodes represent atomic manual or automated activities or sub processes (representing nesting) that must be performed to satisfy the underlying business process objectives. Coordinator nodes allow us to build control flow structures ($fork, choice, loop$ etc.) to manage the coordination

requirements.

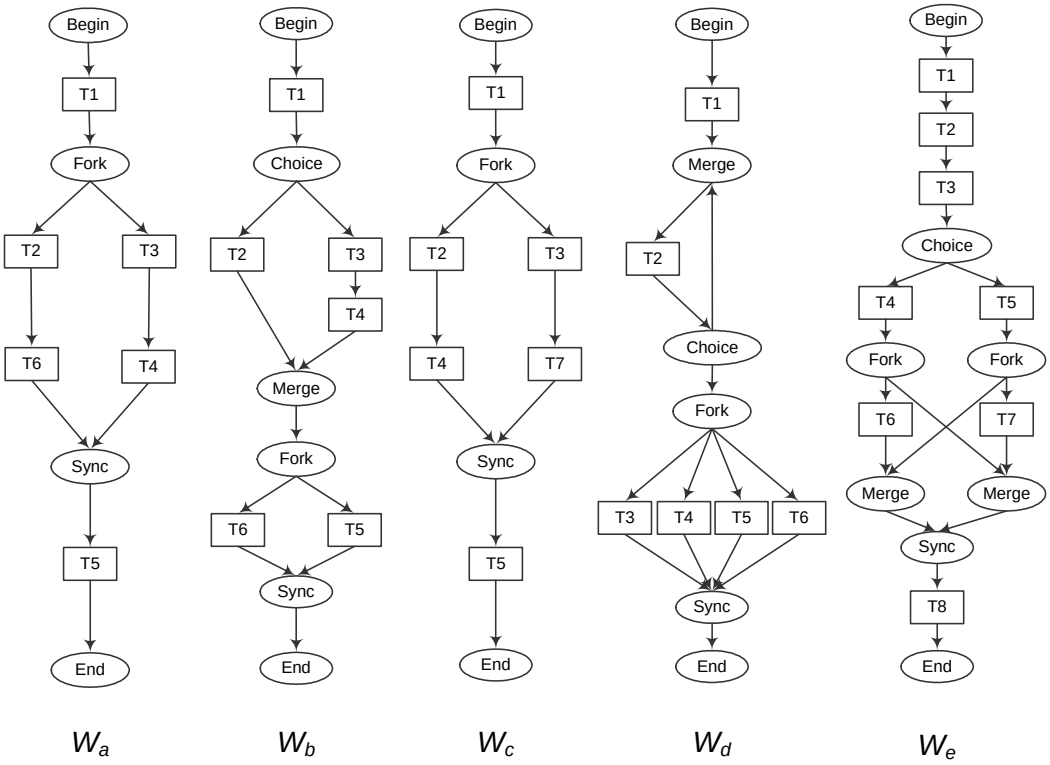


Fig. 12. Example process variant models

An execution sequence of a process variant is referred to as the trace of execution in a process model, which reflects the actual order of task execution at runtime. Typically, a process model with parallel (*fork*) or alternative branches (*choice*) contains more than one possible execution sequences. For example, sequences $\langle T1, T2, T6, T5\rangle$, $\langle T1, T2, T5, T6\rangle$, $\langle T1, T3, T4, T6, T5\rangle$ and $\langle T1, T3, T4, T5, T6\rangle$ are four possible execution sequences in W_b (cf. Fig. 12), since $T2$ and $\langle T3, T4\rangle$ are on alternative branches, and $T5, T6$ are on parallel branches. Note however that for a given process variant, there is exactly one execution sequence resulting from execution.

We follow the general mathematical definition to define an execution sequence: A finite sequence $s = \{s_1, s_2, \ldots, s_n\}$ is a function with the domain $1, 2, \ldots, n$, for some positive integer n. The i-th element of s is denoted by s_i.

Definition 2 (Execution Sequence) *An execution sequence s^W of a process model W is a finite sequence of tasks $T' \subseteq T$ in W, which is defined by the sequence $\langle T_1, T_2, \ldots, T_n\rangle$, $n \geq 1$.*

Note that in some process model W, it is possible that $n > |s^W|$, for some execution sequence $s^W = \langle T1, T2, \ldots, T_n\rangle$. E.g., in W_d, a possible execution sequence is $\langle T1, T2, T2, T3, T4, T5, T6\rangle$, where $T2$ has been executed twice

due to the loop structure. The superscript W of an execution sequence s^W for a process model W can be omitted if no ambiguity is caused.

We now provide a definition for the process variant schema. A retained process variant in PVR is referred to as a *case* and represents the complete design and runtime properties of a variant.

Definition 3 (Process Variant Schema) *A case V is defined by $(id, W, s^W, Res, Dat, T, Com, Mod)$, where*

- *id is the identifier for the process variant V;*
- *W is the process model (N, E) for V defined on the task set $T \subseteq N$;*
- *s^W is the execution sequence of tasks T in V based on W;*
- *$Res = \{Res_1, \ldots, Res_m\}$ is a finite set of resource instances allocated to execute V;*
- *$Dat = \{Dat_1, \ldots, Dat_k\}$ is a finite set of workflow-relevant data items related to V;*
- *$T = \{T_1, \ldots, T_n\}$ is the set of tasks in V, $\forall T_i \in T$, $T_i = (n_i, Res_i, T_i^-, T_i^+)$, where:*
  - *n_i is the identifier of T_i;*
  - *$Res_i \in Res$ is the resource instance allocated to T_i;*
  - *T_i^- and T_i^+ are the time stamps when T_i commenced and completed execution;*
- *Com is an annotation that textually describes the design of the variant;*
- *Mod is the set of modeler(s) who participated in the instance adaptation for V.*

The schema contains instance level features $id, W, s^W, Res, Dat, Com, Mod$ and task level feature T. The id can be combined with the variant symbol V, i.e., V_{10} denotes variant V with the feature $(id, 10)$. Occasionally we omit the subscript i for V when there is no ambiguity. Each element in V is referred to as a *feature* of V. These features can be classified into structural (id, W), behavioral (s^W, Res, Dat) and contextual (Com, Mod) dimension. The process variant repository is the set of all collected process variants, that is $PVR = \{V_1, \ldots, V_n\}$.

PVR is expected to contain a large number of process variants. Table 2 shows some example process variants in PVR based on the graphical process models presented in Fig. 12. Only the variant id, model W and execution sequence s is shown for conciseness. It is likely that many process variants can have the same process model (if the design time template is adapted in the same way), while the execution sequences may still be different.

For example, V_1 and V_8 have the same process model W_a, while due to instance-specific runtime conditions, the execution sequences are different. However, V_1 and V_5 have the same execution sequence although their process models differ.

Table 2
Tabular view of a typical PVR showing the first three features

id	W	s^w	...
V_1	W_a	$\langle T1, T2, T3, T6, T4, T5\rangle$	
V_2	W_c	$\langle T1, T3, T2, T4, T7, T5\rangle$	
V_3	W_d	$\langle T1, T2, T2, T3, T4, T5, T6\rangle$	
V_4	W_b	$\langle T1, T2, T5, T6\rangle$	
V_5	W_d	$\langle T1, T2, T3, T6, T4, T5\rangle$	
V_6	W_e	$\langle T1, T2, T3, T4, T6, T8\rangle$	
V_7	W_e	$\langle T1, T2, T3, T5, T7, T8\rangle$	
V_8	W_a	$\langle T1, T3, T4, T2, T6, T5\rangle$	
...	...	...	...

This observation leads to an interesting problem when defining similarity of process variants regarding W and s, which will be discussed in the next section.

5 Query Processing

PVR uses queries to express requirements for process variant retrieval. Based on different retrieval requirements, a query is a collection of one or more process variant features (cf. Definition 3), describing some desired attributes for the targeted variants. Many of such features can be expressed by a typical structured query language, and can mostly be satisfied using well established query techniques like SQL. For example, in order to find all process variants in which *execution duration is less than 3 hours*, and *any performers of role senior engineer were involved*, a query can be formulated by referring to the time stamp of execution ($|T_i^+ - T_i^-| < 3\ hours$) and the allocated resource instance (Res_i = "senior engineer").

Unlike traditional query systems however, the search criteria for process variants may also include reference to complex structural features. For example, the requirements can be to find all case in which *task Test Hub201 (T2) and Test ExchangeA30 (T3) were executed immediately after Send Test Message (T1)*, and *Reboot Srv59 (T4) was performed in parallel with Test Hub709 (T7)* etc. (cf. W_c in Fig. 12), or simply having the same process model as given in the query.

For queries containing structural features, we propose that the query requirement be expressed in a way similar to the query-by-example (QBE) paradigm,

where a process model W^Q is presented in the query containing the desired structural features, and the objective is to retrieve all cases with a process model W similar to W^Q. W^Q can resemble a complete process model (cf. W_a^Q in Fig. 13), which specifies the exact structure required for the process variants to be retrieved; or a partial process model (cf. W_b^Q in Fig. 13), which contains a fragment of the process model characterizing the desired structural features to be retrieved.

Besides structured feature, a query may also include multi-dimension features found in the process variant schema. For example, *tasks T1, T2 and T3 were performed by a senior engineer in sequence, and finished execution within 1 day* (W_e in Fig. 12), or having execution sequence $\langle T1, T3, T4, T5, T6\rangle$ and *tasks T5 and T6 were in parallel branches in the process model* (cf. W_b in Fig. 12).

It is specifically interesting to investigate providing a facility to find process variants for queries that provide structural criteria, as in the above examples. We now provide a generalised definition for a query in PVR. The following discussions on process variant similarity, however, are focus on the more complex structural features (W and s^W).

Definition 4 (Query) *Let F be the set of all features in PVR. A query Q is defined by the set of query features $\left\{F_1^Q, \ldots, F_n^Q\right\}$, where $\forall F_i^Q \in F$, F_i^Q corresponds to a feature defined in the schema of V. The function $Type$ maps a query feature into one of the process variant features, i.e., $Type : F \mapsto TYPE$, where $TYPE = \left\{id, W, s^W, Res, Dat, T, Com, Mod\right\}$.*

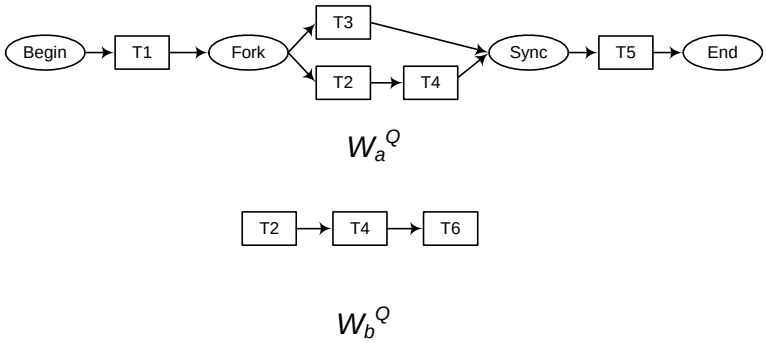


Fig. 13. Example of structural query features, W_a^Q as a complete process model and W_b^Q as a partial process model

5.1 Similarity of Process Variants

In order to determine the degree of match between the desired process variant as described by query features and the potential match, the notion of similarity measure needs to be defined.

Given a query Q with one or more query features $\left\{F_1^Q, \ldots, F_k^Q\right\}$, and a process variant V described by the list of features $\left\{F_1^V, \ldots, F_k^V\right\}$ according to the variant schema (Definition 3), it is desirable that applying a similarity function $Sim(V, Q)$ yields a quantitative figure that indicates the degree of match between V and Q. Nevertheless, each different type of features has specific semantics, e.g., the similarity measure for execution sequence and allocated resources should be different. For each different feature F_i, a similarity function sim should be defined according to the specific semantics of the type of F_i. The overall similarity score $Sim(V, Q)$ is the average of the similarity score for each pair of comparable features (F_i^V, F_i^Q), i.e., $F_i^V, F_i^Q \in F$ and $Type(F_i^V) = Type(F_i^Q)$.

We define the metric space for process variant similarity in the interval of real numbers between 0 and 1, where 0 indicates complete dissimilarity, and 1 indicates complete matching, and a number between 0 and 1 for partial matching. In order to distinguish different similarity functions, the feature type is appended to the function symbol sim, e.g., sim_{F1} denotes the similarity function defined for feature F_1. Note that we refer to $sim(F_i^V, F_i^Q)$ as $sim_{F_i}(V, Q)$

Definition 5 (Generalized Similarity) *Let $Q = \left\{F_1^Q, \ldots, F_k^Q\right\}$, $k \geq 1$, be a query, V be a process variant described by the list of feature $\left\{F_1^V, \ldots, F_k^V\right\}$. Let $sim_{Fi} : F^V \times F^Q \mapsto \{0, \ldots, 1\}$ be a similarity function for a feature type in $TYPE$. Then the overall similarity (Sim) between the process variant V and the query Q is given as:*

$$Sim(V, Q) = \frac{1}{|Q|} \sum_{i=1}^{k} sim_{F_i}(V, Q)$$

wherein (F_i^V, F_i^Q) is a pair of comparable features.

For features in behavioral and contextual dimension (including s, Res, Dat, T, Com and Mod), the similarity function can be defined based on known techniques. For example, simple set membership can be used to compare resources specified in the query to resources utilized in the variant. Similarly, Euclidean distance can be used to define similarity between execution sequences [39]. Case based reasoning has been used to match variants against textual descriptions [36].

5.2 Structural Similarity

The structural feature of process variants is described by a complete or partial process model. Structural aspect is arguably the most important aspect of a

process variant. Defining the similarity based on the metric space is useful for quantifying the degree of match for structural feature, especially for ranking partial matching process models.

Nevertheless, it is argued that graph-based similarity measure alone is inadequate for determining complex matching involving structural features in PVR. This is primarily due to the specialized structural relationships within graphical process models, i.e. models may be structurally different but semantically similar. It is desirable that the similarity of process models can be quantified to some extent. Such that, when the closeness of the query process model and process variant model cannot be visually observed, partial matching variants can be presented using a ranking function to produce a similarity score base on the metric space.

Furthermore, as discussed in previous example (cf. Table 2), it is often the case that exact matching execution sequences may result from different process models. While from the same process model, different execution sequences can be derived. There has been study towards the interplay between the similarity of design time process models and actual execution sequences, which argues for defining structural similarity according to typical execution behaviors as reflected by a chosen set of execution sequences [4]. According to the typical behavior, the more "useful" fragments of the process model are assigned more weight towards the overall structural similarity score.

Based on this observation, it is proposed to define the structural similarity according to both the *structural* and the *execution behavior* of the process model, i.e., the execution sequence. Given the structural feature (as described by a process model) W^Q of a query Q, it is used to retrieve all process variants V in which their process model W is similar to W^Q. Our approach is to first qualify the initial structural matches between a particular W and W^Q, based on structural relationships, where complete and (near perfect) partial matches can be visually identified. A ranking algorithm (similarity function) is then applied for the (not so perfect) partial matching process models to produce a similarity score between each such model and W^Q (presented in Section 5.4). As for the first step, we define three essential structural relationships [34] between W and W^Q:

Definition 6 (Structural Similarity) *Let $W = (N, E)$ be the process model of a process variant V, and $W^Q = (N^Q, E^Q)$ a query process model.*

- *W is said to be structurally equivalent to W^Q if $N = N^Q$ and $E = E^Q$;*
- *W is said to structurally subsume W^Q if $N^Q \in N$, and W^Q preserves the structural constraints between nodes N^Q as specified in W;*
- *W is said to structurally imply W^Q if $N^Q = N$, and W^Q preserves the structural constraints between nodes N^Q as specified in W.*

Additionally, if W and W^Q conform to equivalent relationship, they also conform to *subsume* and *imply* relationship. Given a query process model W^Q, a variant process model variant W is said to be a complete match to W^Q if *equivalent* or *subsume* relationship holds between W and W^Q (*Imply* relationship holds means near prefect partial matches.). The technique to determine complete match is by SELECTIVE-REDUCE [21], which applies graph reduction techniques to determine the match between W and W^Q. The rationale of the technique is to firstly eliminate from N all task nodes that are not contained in N^Q, and secondly to reduce redundant flow relations in E using reduction rules.

The algorithm is provided in appendix A for completeness. Fig. 14 shows the result of applying SELECTIVE-REDUCE to process variant models W_a to W_e (cf. Fig. 12) according to structural query process model W_a^Q (cf. Fig. 13). The reduced process models consist of only tasks $\{T1, T2, T3, T4, T5\}$ as in W^Q. In Fig. 14, the reduced process model RW_c from W_c is structurally equivalent to W_a^Q, which is considered as a complete match.

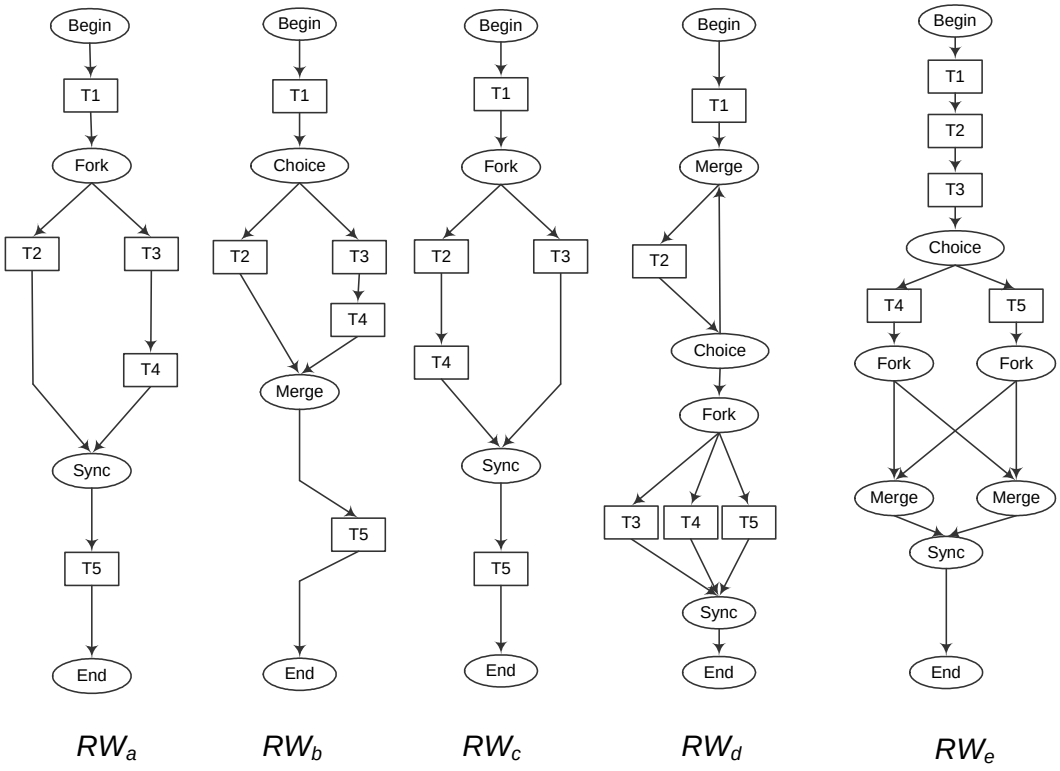


Fig. 14. Reduced process models against query process model W_a^Q

In the rest of this section, the progressive refinement approach for query execution including the ranking technique for partial matches is presented.

In the query processing approach, given a query $Q = \left\{F_1^Q, \ldots, F_k^Q\right\}$, a candidate set of process variants $CV^Q = \{V_1, V_2, \ldots, V_m\}$ is first chosen from PVR, where each $V_i \in CV^Q$ is described by a set of corresponding features $\left\{F_1^{V_i}, \ldots, F_k^{V_i}\right\}$. When a query feature $F_j^Q \in Q$ is to be compared, all $V_i \in CV^Q$ are collected according to the value of feature $F_j^{V_i}$ that is comparable to F_j^Q. Each different $F_j^{V_i}$ is then compared with F_j^Q. For all V_i where $F_j^{V_i}$ is a complete match to F_j^Q, V_i will remain in the candidate set CV^Q. While for those containing partial matching features can be ranked according to the similarity score $sim_{F_j}(V_i, Q)$. The process variants with "the most similar" partial matching feature can also remain in CV^Q. The process variants "not similar enough" are removed from CV^Q. This process is repeated until all $F_j^Q \in Q$ have been compared, or the ideal result set is obtained. The overall similarity score can be calculated for each process variant V_i in the result set by applying $Sim(V_i, Q)$. Fig. 15 provides an illustration for this approach.

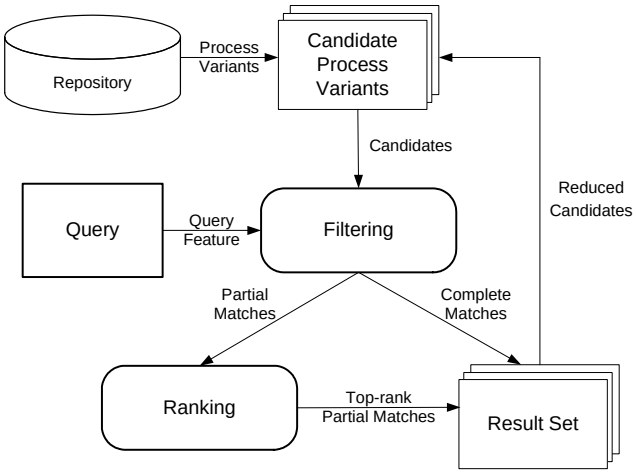


Fig. 15. Progressive-refinement query processing approach

For simple matching features, applying similarity function *sim* produces a ranking for partial matches. The filtering step for complex matching involving structural features however, is to apply SELECTIVE-REDUCE (Section 5.2), which qualifies the structural relationship between the reduced process variant models and the query process model, when complete matches (*equivalent* or *subsume*) and near perfect partial matches (*imply*) can be identified. The ranking step is to provide measurable result that fits in the metric space for ranking partial matches.

As defined previously, the structural feature of a process variant is represented by its process model. In order to produce a quantitative measure for ranking partial matching variants, we take into consideration both the model similarity and the execution behaviour of the models. To evaluate two given process models, we take a list of typical task execution sequences, and evaluate how similar the two models behave in terms of the possibility to produce the whole or partial sequences as in the list. The more sequences the two models support in common, the more similar they are. As a result, a similarity measure is produced. We complete the ranking procedure when this comparison has been repeated between the query process model and each different process model among the partial matching process variants. Which means, the process variants sharing the same process model will be assigned the same ranking.

In particular, the ranking technique, called RANK-STRUCTURAL, calculates the structural similarity between a reduced variant process model RW and the query process model W^Q, with reference to a set of execution sequences S. When given the candidate set of process variants CV^Q containing reduced partial matching models RW, the set of all execution sequences S is the collection of all different execution sequences (s^W) from each process variant V in CV^Q [2]. Each sequence $s \in S$ is calibrated with the number of times it appears or appearance $count(s)$ in CV^Q, and is collectively denoted by Δ (cf. Table 3). The algorithm repetitively compares the two models according to how well each different execution sequence fits in both models. In this way, applying RANK-STRUCTURAL for RW and W^Q produces a relative similarity score with regard to the rest of reduced variant models RW in CV^Q. After applying this algorithm to each different process model in CV^Q, the ranking for each partial matching process variant can be produced.

The algorithm as shown in Fig. 16 takes as inputs a reduced process variant model RW, the query process model W^Q, and Δ, and produces a similarity score sim between RW and W^Q with reference to Δ. Given a process model W and a task $T_i \in T$:

- $Trigger(W, T_i)$ denotes the set of tasks that can be triggered by task T_i in W as the result of execution. E.g., $Trigger(W_e, T1) = \{T2\}$ (cf. Fig. 12). For tasks followed by a *fork* or a *choice* coordinator, it is considered that all subsequent tasks after the coordinator can be triggered. E.g., $Trigger(W_a, T1) = Trigger(W_b, T1) = \{T2, T3\}$;
- $Disable(W, T_i)$ denotes the set of tasks disabled as the consequence of executing T_i, which is defined to realize the semantics of the Choice coordinator.

[2] Note: the actual execution sequence of a process variant is a feature defined in the variant schema, cf. Definition 3.

For example, $Disable(W_b, T2) = \{T3\}$ and $Disable(W_b, T3) = \{T2\}$, which means either $T2$ or $T3$ is executed but not both.

Procedure RANK-STRUCTURAL

Input: RW,W^Q, Δ

Output: sim

Method:

1. $sim, counter \leftarrow 0$
2. $TW, TQ \leftarrow \emptyset$
3. **for** each different sequence s in Δ
4. **for** each task T_j in s, $j \leftarrow 0, \ldots, |s-1|$
5. $TW \leftarrow (TW - \{T_j\} - Disable(RW, T_j)) \cup Trigger(RW, T_j)$
6. $TQ \leftarrow (TQ - \{T_j\} - Disable(W^Q, T_j)) \cup Trigger(W^Q, T_j)$
7. $counter \leftarrow counter + \left(\frac{|TW \cap TQ|}{|TQ|}\right)$
8. $sim \leftarrow sim + \left(\frac{count(s)}{|s|} \times counter\right)$
9. $counter \leftarrow 0$
10. **return** $sim \leftarrow \left(\frac{sim}{|\sum count(s)|}\right)$

Fig. 16. RANK-STRUCTURAL

The algorithm repetitively takes a unique sequence s from Δ for comparison (step 3). Then for every task T_j in each different sequence s (step 4), TW is given the current set of triggered tasks as the result of executing task T_j in RW (step 5). Similarly, TQ is given the current set of triggered tasks as the result of executing task T_j in W^Q (step 6). For each task T_j in s, the proportion of tasks in W^Q triggered by T_j, which are also triggered by T_j in RW is accumulated (step 7). Next the similarity score (resulting from *counter*) for RW and W^Q is accumulated in each sequence s, which is weighted by the number of appearance of s in CV^Q divided by the length of s (step 8). After all different sequences in Δ have been accounted for, the final similarity score is scaled according to the total number of sequences in Δ and returned. Note that $|TQ|$ may evaluate to 0 (step 10). It is postulated that in such case $\frac{0}{0} = 0$, and $\frac{1}{0} = 0$.

Discussion: The algorithm is adapted from the so-called behavioural precision and recall approach from [4] for ranking partial structural matches. In our case, we use the reduced process models as the structural feature to compare the process variant similarity. The list of common execution sequences that

is extracted from the reduced process models is used as a benchmark for the comparison. The complexity of the algorithm is bound by the number of tasks in W^Q and the number of different sequences in Δ. Note that the structural comparison is built upon a native model definition (cf. Definition 1). However, it can be naturally extended to support other graphical model definitions such as BPMN [29] with exclusive gateways (simple merge and join) and parallel gateways (parallel split and join). Furthermore, the algorithm supports multiple control flow patterns in the process model, including sequence and parallel execution, as well as arbitrary loops. This is because 1) any execution of arbitrary loop in the process variant model is reflected by the logged execution sequence, and 2) the algorithm compares the two models by trying to reproduce the same sequence on both models and check for the degree of overlap. Hence, in general cases similarity is higher if both models contain a similar loop structure in the appropriate location, which would reproduce similar execution sequences with repeated tasks. Note that the measure is asymmetric, i.e., $sim(RW, W^Q) \neq sim(W^Q, RW)$.

5.5 Example

Suppose it is required to retrieve process variants that any performer of role *senior engineer* was involved in executing a process model similar to W_a^Q, and its execution duration is less than *3 hours*. A query $Q = \left\{F_1^Q, F_2^Q, F_3^Q\right\}$ containing multi-dimension features is formulated. F_1^Q=$\{$*senior engineer*$\}$ is the resource feature. $F_2^Q = \{< 3\ hours\}$ is the temporal feature derived from the task level features in T. Lastly, $F_3^Q = W_a^Q$ is the structural feature of Q, as defined by the query user.

The initial candidate set $CV^Q = \{V_1, V_2, \ldots, V_m\}$ is first chosen from PVR. $\forall V_i \in CV^Q$, $V_i = \left\{F_1^{V_i}, F_2^{V_i}, F_3^{V_i}\right\}$, where $F_1^{V_i} = role(Vi)$, $F_2^{V_i} = duration(Vi)$, and $F_3^{V_i} = W_i$. *role* is a function defined to extract the roles of performers in Res for a given process variant (case) V. *duration* is a function giving the execution duration of V. These functions can be defined in an application-specific way, e.g., $duration(V) = |T_n^+ - T_l^-|$, where T_n^+ is the completion time stamp of the last task executed in W of V and T_l^- is the start time stamp of the first task in its execution sequence. It can start filtering process variants in CV^Q by $F_1^{V_i}$. Applying $sim_{F_1}(V_i, Q)$ for each $V_i \in CV^Q$ the set of complete matching variants can be identified, i.e., $sim_{F_1}(V_i, Q) = 1$ if $F_1^{V_i}$=F_1^Q=*senior engineer*. As we are only interested in exact matches in F_1^Q and F_2^Q, CV^Q is updated with the set of process variants having complete matching feature $F_1^{V_i}$ (when all partial matching variants are removed from CV^Q). Similarly, CV^Q is further filtered by applying $sim_{F_2}(V_i, Q)$ for each remaining $V_i \in CV^Q$. Suppose $|CV^Q| = 150$ after filtering by F_1^Q and F_2^Q, and for all $F_3^{V_i}$ in CV^Q

there are 5 common process models $\{W_a, W_b, W_c, W_d, W_e\}$ as shown in Fig. 12.

Table 3
The list of all execution sequences S and their counters from reduced partial matching process models (Δ)

s	$count(s)$
$\langle T1, T2, T2, T3, T4, T5\rangle$	5
$\langle T1, T2, T3, T4, T5\rangle$	30
$\langle T1, T3, T2, T4, T5\rangle$	25
$\langle T1, T3, T4, T2, T5\rangle$	45
$\langle T1, T2, T3, T4\rangle$	15
$\langle T1, T2, T3, T5\rangle$	10
$\langle T1, T2, T5\rangle$	20

To filter by the structural feature F_3^Q, we first aggregate all V_i in CV^Q according to $F_3^{V_i}$, i.e., W_i (cf. the first two columns in Table 4). Then applying SELECTIVE-REDUCE to each different W_i, yields reduced variant models $\{RW_a, RW_b, RW_c, RW_d, RW_e\}$ (cf. Fig. 14). The equivalent relationship between RW_c and W_a^Q is identified. As a result, for all $V_i \in CV^Q$ where $F_3^{V_i} = W_c$ are complete matches to F_3^Q, and the rest are partial matches. Applying RANK-STRUCTURAL to the partial matches against W_a^Q provides the similarity ranking. The collection of execution sequences S and counters Δ from all W_i in CV^Q is generated as shown in Table 3. In this case S contains 7 different execution sequences, from 150 process variants.

Table 4 shows the structural similarity ranking after applying RANK-STRUCTURAL to the each partial matches W_i against W_a^Q. A pre-defined similarity threshold (e.g., $sim \geq 0.72$) may be set to define the minimal matching score. In this case, for all $V_i \in CV^Q$ where $F_3^{V_i} \in \{W_c, W_a, W_d\}$, e.g., $\{V_2, V_1, V_8, V_3, V_5, \ldots\}$ remain in the final result set CV^Q.

Table 4
Similarity ranking details for reduced partial matching process models against W_a^Q

W	Variant V	structural similarity
W_c	$\{V_2, \ldots\}$	1.00
W_a	$\{V_1, V_8, \ldots\}$	0.90
W_d	$\{V_3, V_5, \ldots\}$	0.73
W_b	$\{V_4, \ldots\}$	0.71
W_e	$\{V_6, V_7 \ldots\}$	0.67

6 Related Work

The requirements for providing flexibility in process models and execution stem from the need for change in business processes, which have been recognised for over a decade [9,16]. Instance level change is regard as the major strength of flexible workflows and has been receiving much attention in recent years. Industrial standard modeling language BPMN [29] provides a construct called ad hoc sub-process to cater for such a requirement. The introduction of flexible components into process models requires the ability of the business process to execute on the basis of a loosely, or partially specified model, where the full specification is made at runtime. An early attempt in this direction is the flexible sequence in the MOBILE approach [18]. Since then, there have been many proposals offering various solutions [3,5,7,8,12,19,27,31,35].

The proposals for supporting instance level changes can be classified into three major classes, namely, *late selection*, *late modeling* and *late composition* [37].

Late selection is the approach that allows for selecting the implementation for a particular process step at run-time either based on predefined rules or user decisions. Worklets [5] is an example of such an approach. A worklet is a discrete process fragment that is designed to handle a specific action (task) in a larger, composite activity (process). An extensible repository (repertoire) containing a number of different worklets is maintained for a worklet-enabled activity, such that at runtime a preferred worklet is contextually chosen to fulfill the activity goal. The selection of worklets is guided by a set of ripple down rules which associates a worklet with a series of instance-specific conditions.

Late modeling is the approach where parts of the process schema have not been defined at design time, but are modeled at runtime for each process instance. For this purpose, placeholder activities are provided, which are modeled and executed during run-time. The modeling of the placeholder activity must be completed before the modeled process fragment can be executed. Pocket of flexibility [35] is an example of such approach. A pocket is a placeholder activity which contains a set of unstructured inner activities. A fundamental feature is specification of build constraints which essentially control the modeling of the unstructured activities. Late modeling starts when the pocket is instantiated. Then a domain expert defines a corresponding process fragment using a restricted set of modeling elements. The inner activities can be modeled to execute in sequence or parallel, as long as the build constraints are not violated. Upon completion of late modeling the newly defined process fragment is instantiated. Pocket of flexibility presents fundamental concepts on a constraint-based or declarative approach for late modeling. A set of basic constraint types has been identified to provide a flexible means for designing a large number of instance-specific process models. The BPCN framework pre-

sented in this paper is its natural extension, which further provides a richer taxonomy of process constraints with rigorous support for verification based on an underlying formal theory.

Recently, there have been further developments in such declarative approached to process modeling [30], which further re-enforces the need for transferring some of the modeling effort to domain experts at runtime.

Late composition is the paradigm where the instance-specific process model is composed at runtime. This is realized by on-the-fly definition of the control flow dependencies between a set of process fragments at runtime time.

The issue of managing business processes as an information resource was first brought into attention by [20], and has become an important and challenging problem in the field of advanced BPM techniques. [20] points out that process models containing constraints, procedures and heuristics of cooperate knowledge should be regarded as intellectual assets of enterprises. The collection of process models are often regarded as the knowledge base for enterprise operations.

Traditionally this corporate knowledge is represented in process models in various cooperate information systems. However, it is quite often nowadays that a large amount of variances are produced during business process execution. Managing such process variants and subsequently reusing the knowledge from the variants needs to be supported explicitly. In many cases, the source of these variants is the system execution log that stores event-based data for traces of different process executions. As a result, various process mining techniques [2] have been proposed, aiming at reconstructing meaningful process models from executional data. The reconstructed process models can then be used to facilitate a range of process redesign and auditing activities such as to compare with the design models such that the runtime behaviors such as exception handling and derivations can be discovered and diagnosed. In addition, more specific techniques have been proposed to represent and utilize change logs [32] which specifically capture the events and conditions of changes and trace of modification to the process model.

The proposed approach is different from the process mining approach, with the emphasis on supporting knowledge acquisition and process discovery in BPM. In particular, it supports the reuse of past instances of process execution to achieve new operational goals in similar situations. Compared to a typical process execution log, the repository in PVR has a richer schema defined to provide an appropriate characterization to describe the preferred work practices represented through process variants, and subsequently facilitate processing queries with complex requirements for process variants retrieval.

An essential concept in process retrieval is the definition of process equiva-

lence, particularly regarding the structural similarity between two given process models. There have been many proposals for defining process equivalence based on single aspect, such as structural similarity [39], or (execution sequence as) behavioral similarity [4], and contextual similarity [36]. One of the important contributions of this paper is to formally address the issue of similarity definition with regard to process variant properties in multiple dimensions.

Similarity continues to pose several challenges in a number of areas such as string matching, document matching, audio/video matching etc. Noteworthy contributions that enable a better understanding of the notion of similarity from a process perspective include [13] where a classification of process differences is presented, [14] where process similarity is studied based on so-called causal footprints, and [15] that tackles the problem from a metadata perspective. Complementary work on process quality [26] is also relevant here, as various model characteristics (e.g. node density, coupling, cohesion etc) impact on the complexity of similarity detection computations. Recently introduced re-factoring [38] of process models may assist in this regard, although further research is warranted before practical solutions can be fully supported.

7 Conclusion and Future Work

Variations in work practice often represent the competitive differentiation within enterprise operations. In this paper we have argued for acknowledging the value of variants in business process management platforms. We have presented how process constraints can be used to express minimal restrictions on the selection and ordering of tasks for variants of the targeted business process. The selection and scheduling constraints are specified at design time, through intuitive constraint notations. With the graphical notations, the constraint specification can be used to provide visual support for domain experts to design compliant process variants during instance adaptation. We have also presented an approach for managing such process variants as an information resource, thus providing a whole-of-cycle solution. The presented methods provide effective means of searching and matching process variants against a given query from simple to complex aspects, and generate result sets that can be conveniently ranked, thereby empowering process designers to tap into effective precedents. In our future work we are planning to implement the query processing approach so that empirical evaluation can be performed and scalability and complexity analysis can be rigorously conducted.

References

[1] van der Aalst, W. M. P., ter. Hofstede, A. H. M., Weske, M.: Business Process Management: A Survey. In. van der Aalst, W. M. P., H. M. ter Hofstede, A., Weske, M. (Eds.): Proc. of International Conference on Business Process Management (2003)

[2] van der Aalst, W. M. P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow Mining: A Survey of Issues and Approaches. Data & Knowledge Engineering, Vol.47 (2003) 237 - 267

[3] van der Aalst, W. M. P., Weske, M.: Case handling: a new paradigm for business process support. Data & Knowledge Engineering, Vol.53(2) (2006) 129-162

[4] van der Aalst, W. M. P., de Medeiros, A.K. Alves., 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.): Proc. of 4th International Conference on Business Process Management (2006)

[5] Adams, M., ter Hofstede, A. H. M., Edmond, D., van der Aalst., W. M. P.: Implementing Dynamic Flexibility in Workflows using Worklets. BPMcenter.org (2006)

[6] Allen, J.: Maintaining knowledge about temporal intervals. Communications of the ACM, vol.26 (1983) 832-843

[7] Berens, P.: The FLOWer Case-Handling Approach: Beyond Workflow Management. In Marlon Dumas, A. H. M. ter. Hofstede., Process-Aware Information Systems. John Wiley & Sons, Inc. (2005) 363-395

[8] Browne, E., Schrefl, M., Warren, J.: Goal-Focused Self-Modifying Workflow in the Healthcare Domain. In Proc. 37th Hawaii International Conference on System Sciences (HICSS-37) (2004)

[9] Casati, F., Sanchez, G., Pernici, B., Pozzi, G., Vonk, J.: Workflow Conceptual Model, Database Support for Workflow Management: the WIDE Project, Kluwer Academics Publishers (1999)

[10] Casati, F.: Industry Trends in Business Process Management: Getting Ready for Prime Time. Proc. DEXA'05 Workshops (2005) 903-907

[11] Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers (2003)

[12] Dias, P., Vieira, P., Rito-Silva, R.: Dynamic Evolution in Workflow Management Systems. In Proc. 14th International Workshop on Database and Expert Systems Applications (DEXA'03) (2003) 254-260

[13] Dijkman, R.: A Classification of Differences between Similar Business Processes. In Proc. 11th IEEE EDOC Conference (EDOC2007) (2007) 37-50

[14] van Dongen, B., Dijkman, M., Mendling, J.: Measuring Similarity between Business Process Models. In Proc. 20th International Conference on Advanced Information Systems Engineering (CAiSE'08) (2008) 450-464

[15] Ehrig, M., Koschmider, A., Oberweis, A.: Measuring similarity between semantic business process models. In Proc. the fourth Asia-Pacific conference on Conceptual modelling, Ballarat, Australia (2007) 71-80

[16] Ellis, C. A., Keddara, K., Rozenberg, G.: Dynamic change within workflow systems. In Proc. COOCS'95 (1995) 10-21

[17] Indulska, M., Chong, S., Bandara, W., Sadiq, S., Rosemann, M.: Major issues in business process management: A vendor perspective. In Proc. the Pacific Asia Conference on Information Systems (PACIS2007) (2007)

[18] Jablonski, S.: MOBILE: A Modular Workflow Model and Architecture. In Proc. Int'l Working Conference on Dynamic Modelling and Information Systems, Nordwijkerhout, 1994 (1994)

[19] Kammer, P., Bolcer, G., Taylor R., Bergman, M.: Techniques for Supporting Dynamic and Adaptive Workflow. In: Computer Supported Cooperative Work (CSCW), Vol.9(3-4) (2000)

[20] Leymann, F., Altenhuber, W.: Managing Business Processes as an Information Resource. IBM Systems Journal, 33(2) (1994)

[21] Lu, R., Sadiq, S.: Managing Process Variants as an Information Resource. In Proc. 4th International Conference on Business Process Management (BPM2006), Vienna, Austria (2006)

[22] Lu, R., Sadiq, S., Padmanabhan, V., Governatori, G.: Using a Temporal Constraint Network for Business Process Execution. In Proc. 17th Australasian Database Conference (ADC2006), Hobart, Australia (2006)

[23] Lu, R., Sadiq, S.: A Reference Architecture for Managing Business Process Variants. In Proc. of 9th International Conference on Enterprise Information Systems (2007)

[24] Lu, R.: Constraint Based Flexible Business Process Management. PhD Thesis, School of Information Technology and Electrical Engineering, The University of Queensland. Awarded 15 May 2008 (2008)

[25] Madhusudan, T., Zhao, L., Marshall, B.: A Case-Based Reasoning Framework for Workflow Model Management. Data Knowledge Engineering, Vol.50(1) (2004) 87-115

[26] Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. PhD Thesis, Vienna University of Economics and Business Administration (2007)

[27] Muller, R., Greiner, U., Rahm, E.: AGENT WORK: a workflow system supporting rule-based workflow adaptation Data Knowl. Eng., Elsevier Science Publishers B. V., Vol.51 (2004) 223-256

[28] OASIS. Business process execution language for web services version 1.1 (bpel4ws 1.1) specification. Standardisation, Organization for the Advancement of Structured Information Standards (OASIS) (2006).

[29] Object Management Group (OMG): Business Process Modeling Notation (BPMN) Specification 1.0 (2006)

[30] Pesic, M., Schonenberg, M. H., Sidorova, N., van der Aalst, W. M. P.: Constraint based workflow models: Change made easy. In Proc. OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007 (2007) 77-94

[31] Reichert, M, Dadam, P.: ADEPTflex-Supporting Dynamic Changes of Workflows Without Losing Control. J. Intell. Inf. Syst. Vol.10(2) (1998) 93-129

[32] Rinderle, S., Reichert, M.: Data-Driven Process Control and Exception Handling in Process Management Systems. In Proc. 18th International Conference on Advanced Information Systems Engineering (2006)

[33] Sadiq, W., Orlowska, M.: Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. In Proc. of 11th International Conference on Advanced Information Systems Engineering (CAiSE'99) (1999) 195-209

[34] Sadiq, W., Orlowska, M.: On Business Process Model Transformations. In Laender, A, Liddle, S., Storey, V. (Eds.): In Proc. of 19th International Conference on Conceptual Modeling (2000) 267-280

[35] Sadiq, S., Sadiq, W., Orlowska, M.: A Framework for Constraint Specification and Validation in Flexible Workflows. Information Systems, Vol.30(5) (2005)

[36] Weber, B., Wild, W., Breu, R.: CBRflow: Enabling adaptive workflow management through conversational case-based reasoning. In Proc. 9th European Conference on Case-Based Reasoning (ECCBR2004) (2004) 434448

[37] Weber, B., Rinderle, S., Reichert, M.: Change Patterns and Change Support Features in Process-Aware Information Systems. In Proc. 19th International Conference on Advanced Information Systems Engineering (CAiSE'07) (2007) 574-588

[38] Weber, B., Reichert, M.: Refactoring Process Models in Large Process Repositories. In Proc. 20th International Conference on Advanced Information Systems Engineering (CAiSE'08) (2008) 124-139

[39] Wombacher, A., Rozie, M.: Evaluation of Workflow Similarity Measures in Service Discovery. In Schoop, M., Huemer, C., Rebstock, M., Bichler, M. (Eds.): Service Oriented Electronic Commerce. LNI, Vol.80. GI (2006)

A SELECTIVE-REDUCE algorithm

The goal of the original algorithm in [33] is to reduce a process graph into an empty graph in order to verify structural correctness. In our approach, the algorithm is modified to reduce a variant that has an equivalent or subsume

relationship with the query, into a structurally identical graph (not empty) as the query. A detail description of the original algorithm can be found in [33].

The following terms and functions will be used to present the method. For a given process model $W(N, F)$:

- $size[W] = size[N] + size[F]$ represents the total number of nodes (N) and flows (F) in W.

For each flow $f \in F$, following basic attributes are defined:

- $fromNode[f] = n$ where $n \in N$ represents from node of f.
- $toNode[f] = n$ where $n \in N$ represents to node of f.

For each node $n \in N$, following basic attributes are defined:

- $nodeType[n] \in \{task, coordinator\}$ represents type of n.
- $coordinatorType[n] \in \{fork, synchronizer\}$.
- $dout[n] =$ out degree of n, i.e., number of outgoing flows from n.
- $din[n] =$ in degree of n, i.e., number of incoming flows to n.
- $OutFlows[n] = \{f : f \in F, fromNode[f] = n\}$, i.e. the set of outgoing flows from n.
- $InFlows[n] = \{f : f \in F, toNode[f] = n\}$, i.e. the set of incoming flows to n.
- $OutNodes[n] = \{m : m \in N, f \in F, fromNode[f] = n, toNode[f] = m\}$, i.e. the set of succeeding nodes that are adjacent to n.
- $InNodes[n] = \{m : m \in N, f \in F, toNode[f] = n, fromNode[f] = m\}$, i.e. the set of preceding nodes that are adjacent to n.
- $F[W] = \{f : f \in W\}$, i.e. the set of flows in process model W.
- $N[W] = \{n : n \in N\}$, i.e. the set of nodes in process model W.
- *delete* n is a procedure that removes n from $N[W]$ and the set of outgoing flows $OutFlows[n]$ from n.

The algorithm is presented as follows:

Procedure SELECTIVE-REDUCE

Input: Process model W, query graph Q

Output: Reduced process model RW

Method:

if $N[Q] \subseteq N[W]$ **then**

$lastsize \leftarrow size[W] + 1$

while $lastsize > size[W]$ **do**

$lastsize \leftarrow size[W]$

/*Terminal-reduces first and last non-query task nodes*/

for each node $n \in N[W]$, $n \notin N[Q]$, **do**

if $din[n] + dout[n] \leq 1$ **then**

delete n

/*Sequential-reduces nodes with one incoming and one outgoing flow*/

else if $din[n] = 1$ and $dout[n] = 1$ **then**

$toNode[top[InFlows[n]]] \leftarrow top[OutNodes[n]]]$

delete n

/*Adjacent - merges adjacent forks or syncs*/

else if $din[n] = 1$ and $dout[n] > 1$ and $nodeType[n] = nodeType[top[InNodes[n]]]$

then

for each transition $f \in OutFlows[n]$ **do**

$fromNode[f] \leftarrow top[InNodes[n]]$

delete n

else if $dout[n] = 1$ and $din[n] > 1$ and $nodeType[n] = nodeType[top[OutNodes[n]]]$

then

for each transition $f \in InFlows[n]$ **do**

$toNode[f] \leftarrow top[OutNodes[n]]$

delete n

end if

end while

/*Closed - reduces redundant flow from fork to sync*/

if $lastsize = size[W]$ **then**

for each node $n \in N[W]$, $n \notin [Q]$, **do**

if $nodeType[n] = fork$ and $dout[n] > 1$ **then**

$NodeSet \leftarrow \emptyset$

for each transition $f \in OutFlows[n]$ **do**

if $nodeType[toNode[f]] = synchronizer$ **then**

if $toNode[f] \notin NodeSet$ **then**

$NodeSet \leftarrow NodeSet \cup \{toNode[f]\}$

else

delete f

end if

end if

end if

end if

end if