SCHOOL OF INFORMATION TECHNOLOGY
& ELECTRICAL ENGINEERING
THE UNIVERSITY OF QUEENSLAND
Brisbane
Queensland 4072
Australia

Phone: +61 7 3365 1009
Fax: +61 7 3365 4999
Email: tricia@itee.uq.edu.au

Technical Report
No. 464

On Managing Process Variants as an Information Resource

Ruopeng Lu
Shazia Sadiq

June 2006

On Managing Process Variants as an Information Resource

Ruopeng Lu, Shazia Sadiq

School of Information Technology and Electrical Engineering,
The University of Queensland, Brisbane, QLD, 4072, Australia
{ruopeng, shazia}@itee.uq.edu.au

Abstract. Many business solutions provide best practice process templates, both generic as well as for specific industry sectors. However, it is often the variance from template solutions that provide organizations with intellectual capital and competitive differentiation. Although variance must comply with various contractual, regulatory and operational constraints, it is still an important information resource, representing preferred work practices. In this paper, we present a modeling framework that is conducive to constrained variance, by supporting user driven process adaptations. The focus of the paper is on providing a means of utilizing the adaptations effectively for process improvement through effective management of the process variants repository (PVR). In particular, we will provide deliberations towards a facility to provide query functionality for PVR that is specifically targeted for effective search and retrieval of process variants.

Keywords: business process modeling and analysis, business process and e-service repositories.

1 Introduction

The merits and limitations of prescriptive approaches to process modeling have been widely debated in BPM literature [1, 2]. Whereas the need to provide flexible frameworks for process modeling and deployment is well understood, many challenges still remain in the provisioning of such systems. Especially, the ability to provide generic solutions that delivers on both (often contradicting) requirements of strategic control and operational flexibility. It is evident 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. This diversity needs to be both encouraged and controlled.

A major difficulty in this issue lies in the fact that the requisite knowledge, that drives the diverse practices at an operational level, is only tacitly available, i.e., it is not found in corporate manuals, or policy documents. This knowledge constitutes the corporate skill base and is found in the experiences and practices of individual workers, who are domain experts in a particular aspect of the overall operations. There is significant evidence in literature on the difficulties in mapping process logic to process models [3, 4]. With the absence of explicit articulation, the complexity is increased manifold.

Furthermore, industry studies [5] show that a substantial percentage of the business process management efforts go into defining the process requirements, while the process modellers are typically business owners rather than domain experts. We believe that this is a limitation in current solutions, and part of the modelling effort needs to be transferred to domain experts who make design decisions based on (1) their expertise and (2) case specific conditions.

In this paper, we utilize a framework for process modelling and deployment [6, 7] that harnesses successful work practice and provides the ability to build a valuable information resource from them. The framework consists of two major components: (1) A constraint-based process modelling approach, namely Business Process Constraint Network (BPCN); and (2) a repository for case specific process models, called process variant repository (PVR). BPCN provides a platform for dynamic process modelling and execution. While PVR provides a well-formed structure to store past process designs, as well as an instrument to utilize process variants as an information resource.

It is the last aspect which forms the focus of this paper. PVR can be expected to build into a high volume repository. Additionally, each process variant is a complex object by itself. Structuring and subsequently querying such a large and complex data source requires careful planning and design. The aim of this paper is to provide an effective approach for structuring and querying PVR.

The rest of the paper is organized as follows. Section 2 will provide the background concepts for BPCN and PVR. Section 3 discusses in detail the approach on managing and querying PVR. Related work is presented in section 4, and conclusions drawn from this work in section 5.

2 Background

Consider the following scenario of customer complaint management in a telecommunication company. Inquiry logging and reporting procedures are predictable and repetitive. However, response to individual complaints and subsequent diagnostic tests performed are prescribed uniquely for each case, but nonetheless have to be coordinated and controlled. Suppose that a number of diagnostic tests, (say 8 tests, *T1, T2,…,T8*), are available. Any number of these tests can be prescribed for a given request, in some preferred order (cf. Fig.1). The supervising engineer has the flexibility to design a plan that best suits the customer request. The knowledge that guides the design of inquiry response is implicit and owned by domain experts. Most often such decisions can only be made based on case specific conditions that cannot be fully anticipated at design time.

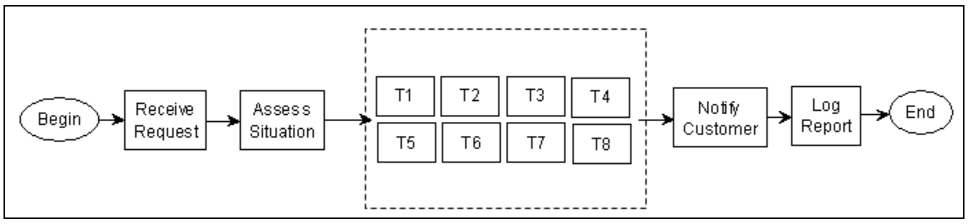

Fig. 1. Example business process

This is an example typical of the need for operational flexibility raised earlier. In the backdrop of this scenario, we briefly introduce a constraint based process modeling approach that specifically targets such requirements. The approach based on constraint specification, is simple and has minimal impact on the underlying process management system. We assume that the underlying system supports a typical graph-based process model and a state-based execution model. Such process models support typical constructs like 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.

Let W represent such a **process model**, where W (N, F) is defined through a directed graph consisting N: Finite Set of Nodes, F: Flow Relation $F \subseteq N \times N$. Nodes are classified into tasks (T) and coordinators (C), where $C \cup T$, $C \cap T = \emptyset$. Task nodes represent atomic manual / automated activities or sub processes 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.

In the proposed approach, we introduce the notion of a **process template** PT defined by < P, C> where P is a **set of process components**, for all $p \in P$, p is a sub-process of type W, or a single task; and C is a **set of constraints** that defines relations between the (properties of) process components.

For example, in the customer complaints management process, P = {*T1, T2, T3, T4, T5, T6, T7, T8}* and there can be a number of operational constraints defined on the diagnostic tests, such as T1 must be performed before *T5*; and *T2* and *T4* must be done in parallel.

A process template PT thus defines the *minimal* constraints required for the business process to provide necessary enforcement. This approach is supported by a runtime environment [7], wherein the template is concretized by designated domain experts (e.g. senior-engineer) using runtime instance specific knowledge. Thus there will be a mapping from $PT \rightarrow W$, where one template may map to several process models. In order for W to be a valid model, all constraints $c_1, c_2, \ldots, c_n \in C$ as given in $PT = <P, C>$ must hold in W, that is $c_1 \wedge c_2 \wedge \ldots \wedge c_n$ must be true. Execution of W can then be provided through a typical state-based process execution engine.

The main feature of this approach is the utilization of the constraint set C. We identify three essential classes of constraints:

1. **selection** constraints that define *what* activities constitute the process,
2. **scheduling** constraints that define *when* these activities are to be performed, both in terms of ordering as well as temporal dependencies, and lastly
3. **resource** constraints that define *which* resources are required to perform the activities.

In previous work [6, 7], we developed a number of constraint types under these classes. For example *serial* is a type of scheduling constraint, where given activities must be executed serially, i.e., not concurrently. However the choice of order remains flexible and is determined by the user during runtime instance customization. A practical example of a serial constraint can be found in healthcare. Pathologies and medical imaging labs need to schedule a large number of tests in different departments. A number of tests can be prescribed for a given patient, e.g., blood test, X-Ray, ECG. These tests can be done in any order but only one at a time.

Business Process Constraint Network (BPCN) [6] has been developed to provide formal underpinning to the notion of process templates. It provides the ability to accept a set of various constraint types; and provides methods for checking constraint network consistency. These details are not included in this paper, but in the next section, we will utilize BPCN concepts as background. The BPCN is essentially a design approach, but is supported by an execution environment in which process instances can be tailored to case specific needs (e.g., a particular configuration of tests prescribed by a service plan). We have referred to these individually tailored process instances as **process variants**, each of which represent the preferred work practice, but are also valid in terms of process constraints as defined by the BPCN. Conformance to constraints is guaranteed through constraint validation algorithms [7]. This execution environment allows the generation of potentially a large number of process variants, which are stored in the process variant repository (PVR).

Although all process variants satisfy the same set of constraints, they may vary significantly. Over time, the repository of such process variants can build into an immense corporate resource. We argue that such a resource can provide valuable insight into work practice, help externalize previously tacit knowledge, and provide valuable feedback on subsequent process design and improvement.

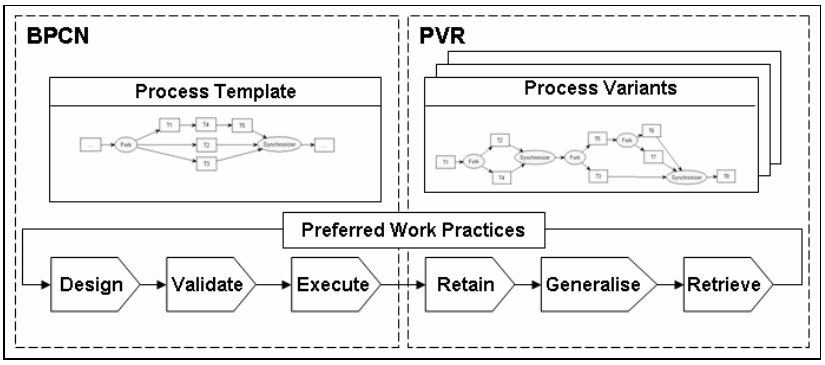

Fig. 2. Framework overview

It is important to note that the way that domain experts reason about the situation during process modeling cannot be truly reconstructed using computational techniques. However, the purpose of the process repository is to capture, structure and subsequently extract the decisions that led to a particular design. These design decisions are embedded in various process properties e.g., process data values. If a significant number of process variants are designed in a similar way, with similar process properties, then this is an indication of a preferred (and likely successful) work practice (cf. Fig.2).

It is evident that providing functions to manage such a resource can be highly beneficial. However, the repository consists of very complex objects, i.e. process variants. Effectively structuring and efficiently querying these complex objects is very challenging. The following section presents our approach to tackle this problem.

3 Repository for Process Variants

The fundamental goal of PVR is to provide appropriate characterization to describe the preferred work practices represented through process variants, and subsequently generalize the conditions contributing to the preference. At any time, the repository may be queried for **precedents under specified criteria**. The first question towards managing the repository is how to appropriately structure it such that subsequent querying can be satisfactorily provided. The second question is how can a query be formalized and processed. This section presents the solutions to these problems.

3.1 Repository Schema

When a process instance completes execution, the model corresponding to the process variant as well as essential execution properties are stored in the PVR. The schema of the repository defines the structure according to which process variants are stored. Ideally, the schema should include all information taken into account during design. However, there are limitations to this effect since the thought process of the designer cannot be captured.

We identify at least three types of information that should be used to describe process variants:

1. **Annotative** – Annotative descriptions can be very useful when descriptions are to be scanned by humans. There are reports in literature [8] that endorse the use of textual descriptions for workflow cases.
2. **Operational** – Operational descriptions relate to execution properties of process variants such as total time of execution, resources utilized etc.

3. **Structural** – Structural descriptions relate to the task set and behavior (control dependencies) of the process variant.

Confining description of process variants to essential operational and structural aspects, we can define a process variant *V* by <*Id*, *Desc*, *R*, *D*, *T*, *W*>, where

- *Id* is the identifier for the process variant;
- *Desc* is an annotation that textually describes the design of the variant;
- $R = \{R_1,\ldots,R_k\}$ is a finite set of resource instances allocated to *V*;
- $D = \{D_1,\ldots,D_l\}$ is a finite set of process-relevant data items related to *V*;
- $T = \{T_1,\ldots,T_m\}$ is a finite set of tasks. $\forall t_i \in T$, $t_i = <n_i, r_i, t_i^-, t_i^+>$, where n_i is the name of task t_i; $r_i \in R$ is the resource instance allocated to task t_i; t_i^- and t_i^+ are the times when task t_i started and finished execution. Execution duration dur_i of t_i is given by $dur_i = | t_i^+ - t_i^- |$;
- *W* represents the process model (*N*, *F*) for *V* defined on the task set $T \subseteq N$. Since *W* represents a particular instance, coordinator nodes types are limited, i.e., $\forall$ n $\in$ *C*, *CoordType*: n $\rightarrow$ {*fork, synchronize, begin, end*}. Coordinator types have typical semantics, as summarized below:
  - $\forall n \in N$, din: N $\rightarrow$ Nat, $din(n)$: Number of incoming flows for node *n*.
  - $\forall n \in N$, $dout$: N $\rightarrow$ Nat, $dout(n)$: Number of outgoing flows for node *n*.
  - $\forall n \in C$, $din(n) \geq 2 \vee dout(n) \geq 2$ where $n \neq begin$ and $n \neq end$.
  - $\forall n \in T$, $din(n) \leq 1 \vee dout(n) \leq 1$

The fork coordinator represents an *AND-SPLIT* and facilitates the concurrent triggering of all nodes on its outgoing flows, and will have $dout(n) \geq 2$. The synchronizer pairs with the fork, and represents an *AND-JOIN*. The synchronizer coordinator waits for all incoming flows to be triggered, before allowing the control flow to progress further and will have $din(n) \geq 2$.

The process variant repository is the set of all variants defined (over a period of time) for a particular process template *PT*, that is $PVR = \{V_1,\ldots,V_n\}$.

3.2 Query Facility

As the number of entries in PVR grows, effective management of the repository will become a challenging issue, due to the complexity of the data required to describe the variants. A **query** is a statement of information needs, which is formulated according to one or more aspects of process variants. Querying is the process of utilizing captured cooperate knowledge when executed process variants are retrieved from the repository to assist process redesign or analysis. Query requirements for PVR can be wide ranging. Essentially, PVR queries require precedent searching under specified criteria**.** Examples of search criteria can be conditions such as find all process variants with *an execution duration of not more than 3 days*, or where *no performers of role senior engineer* were involved. Such queries can mostly be satisfied using well established techniques. Unlike traditional query systems however, the search criteria may also include reference to complex objects, e.g., *find all process variants in which tests T1, T3 and T6 were performed in parallel*.

We are specifically interested in providing a facility to find process variants for queries that provide complex (structural) criteria, as in the above example. As introduced previously, the structural aspect of process variants is represented through *W* (a directed graph with particular node semantics). The issue here is how to characterize the structural features of process variants such that queries that require searching the repository on the structural aspect can be efficiently satisfied.

As an example, consider the following collection of process variants for the scenario introduced in section 2. Details of variants which are common for all (e.g., begin/end, tasks “Receive Request”, “Assess Situation”, “Notify Customer” and “Log Report” in Fig.1) have been omitted for clarity. All process variants in Fig.3 satisfy same constraints, which are *T1* must be performed before *T5*; *T2* and *T4* must be done in parallel. PVR can be expected to contain hundreds if not thousands of such variants for a given process.

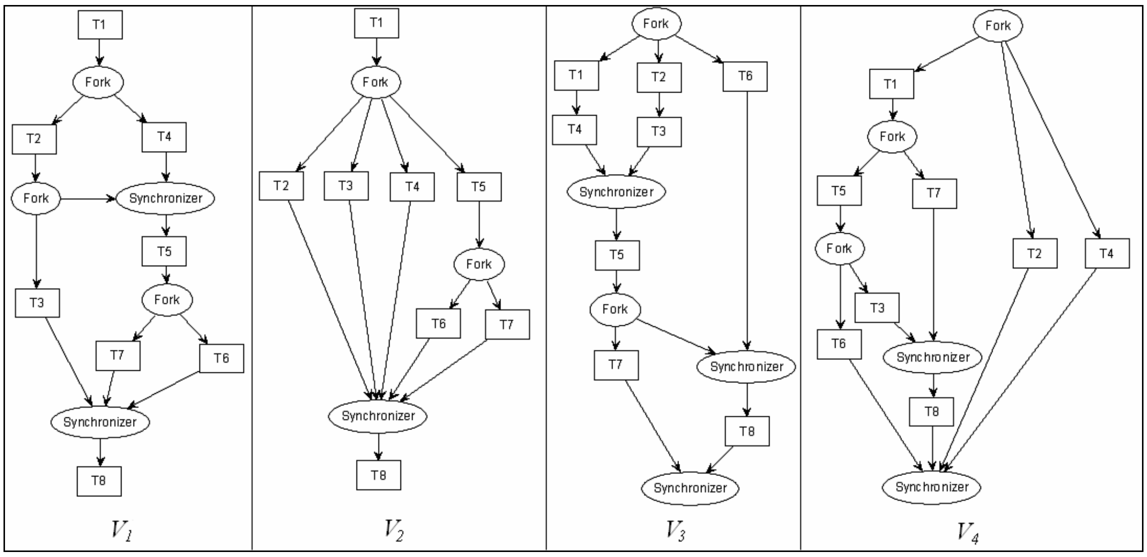

Fig. 3. Example Process Variants V_1, V_2, V_3 and V_4

The foremost question to be addressed in this regard is *how to specify* a query on structural aspect of the variants. The second, but just as important question is how to *search, match and retrieve* efficiently the corresponding variants that satisfy the query criteria.

We propose to define queries on structural aspect as process fragments, defined using a subset of the graphical language used in *W* (begin and end coordinator nodes are precluded). As an example consider the queries illustrated in Fig.4.

Let *Q* be the process (sub)graph that represents a query, i.e. the criteria for selection of process variants. The structure of *Q* can be “similar” to structural definition of a variant V_k, but may not be identical. We define similarity between *Q* and V_k through two relationships, namely equivalent and subsume [9].

A process graph *P (N, F)* is said to be structurally **equivalent** to a process graph *P’(N’, F’)* if node sets *N* and *N’* as well as flow relations *F* and *F’* of *P* and *P*’ respectively, are the same.

A process graph *P (N, F)* is said to structurally **subsumes** a process graph *P’(N’, F’)* if *N’ Í N*, and *P’* preserves the structural constraints between nodes *N’* as specified in *P*. Additionally, if *P* and *P’* conform to equivalent relationship, then they also conform to subsume relationship.

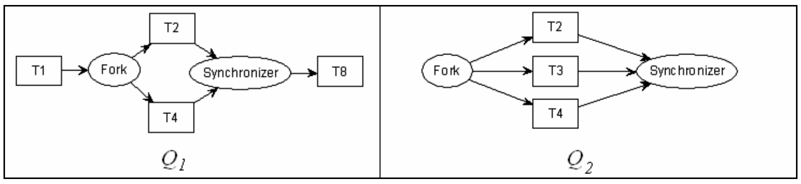

Fig. 4. Example Queries Q_1 and Q_2

For example, query graph Q_1 is subsumed by process variants V_1 and V_2 as given in Fig.3. The structural constraints in Q_1 indicate that tasks *T1* and *T8* are executed in order, and *T2* and *T4* in a fork structure after *T1* and before *T8*. Both V_1 and V_2 contain *T1, T2, T4* and *T8*, as well as preserve the structural constraints between them.

In order to determine whether a given variant is in an equivalent or subsume relationship with a specified query, we propose a matching method `SELECTIVE_REDUCE`, which uses graph reduction techniques to determine the match. The method is assumed to be executed on only those variants from PVR where the node set of the variant is a superset of the node set of a specified query. The basic intuition behind `SELECTIVE_REDUCE` is to firstly eliminate from the node set of the variant all task nodes that are not contained in the node set of the query, and secondly to reduce the flow relation using three reduction rules [9], namely *sequential*, *adjacent* and *closed*. Fig.5 illustrates the applications of these reduction rules, where the solid rectangles represent the relevant tasks required by the query and the hollow rectangles the irrelevant tasks. The goal of the original algorithm in [9] 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. In [10], a detail description of the algorithm can be found.

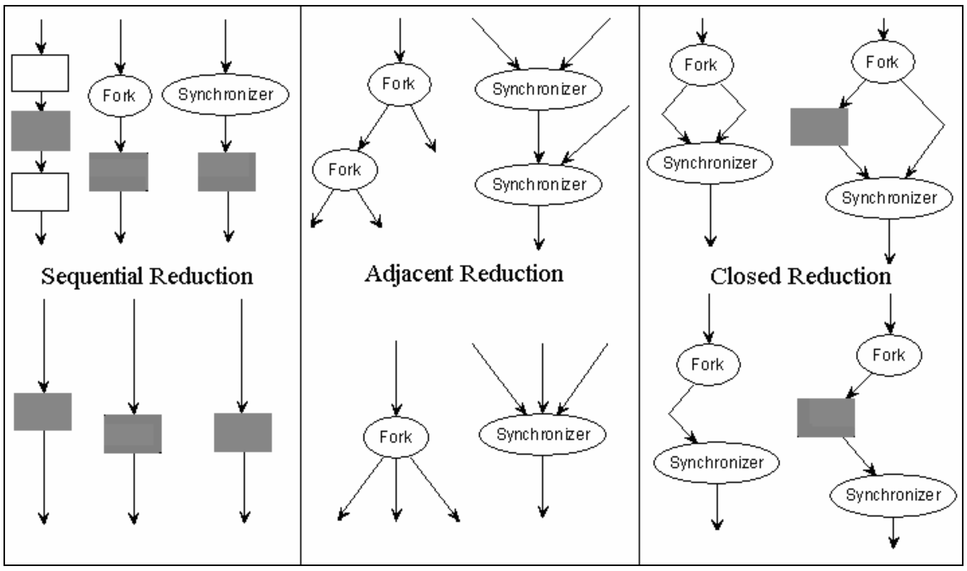

Fig. 5. Sequential, Adjacent and Closed Reduction Rules

The following terms and functions will be used to present the method. For a given process graph *P* (*N*, *F*):

– *size*[*P*] = *size*[*N*] + *size*[*F*] represents the total number of nodes (*N*) and flows (*F*) in *P*.

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*] ∈ { *task, coordinator* } represents type of *n*.
– *coordinatorType*[*n*]∈ {*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* ∈ *F* and *fromNode*[*f*] = *n* }, i.e. the set of outgoing flows from *n*.
– *InFlows*[*n*] = {*f* : *f* ∈ *F* and *toNode*[*f*] = *n* }, i.e. the set of incoming flows to *n*.
– *OutNodes*[*n*] = {*m* : *m* ∈ *N* and ∃ *f* ∈ *F* where *fromNode*[*f*] = *n* and *toNode*[*f*] = *m* }, i.e. the set of succeeding nodes that are adjacent to *n*.
– *InNodes*[*n*] = {*m* : *m* ∈ *N* and ∃ *f* ∈ *F* where *toNode*[*f*] = *n* and *fromNode*[*f*] = *m* }, i.e. the set of preceding nodes that are adjacent to *n*.
– *F*[*P*] = {*f* : *f* ∈ *P*}, i.e. the set of flows in process graph *P*.
– *N*[*P*] = {*n* : *n* ∈ *N*}, i.e. the set of nodes in process graph *P*.
– delete *n* is a procedure that removes *n* from *N*[*P*] and the set of outgoing flows *OutFlows*[*n*] from n.

The algorithm is presented as follows:

```
Algorithm 1 SELECTIVE_REDUCE
  Input process graph P, query graph Q
  Output reduced process graph P

  if $N[Q] \subseteq N[P]$ then
     $lastsize \leftarrow size[P] + 1$
     while $lastsize > size[P]$ do
        $lastsize \leftarrow size[P]$
        /* Terminal reduction – reduces first and last non-query task nodes */
        for each node $n \in N[P]$, $n \notin N[Q]$, do
           if $din[n] + dout[n] = 1$ and then
              delete n
           /* Sequential reduction - 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 reduction - merges adjacent forks or synchronizers */
            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 reduction – reduces redundant flow from fork to synchronizer */
    if $lastsize = size[P]$ then
        for each node $n \in N[P]$, $n \notin N[Q]$, do
            if $nodeType[n] = fork$ and $dout[n] > 1$ then
                $NodeSet \leftarrow \{\}$
                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
```

Fig.6 illustrates the process of reducing the process graph of V_1 by applying SELECTIVE_REDUCE, given $N[Q_1]$ = {*T1, T2, T4, T8*}, (query graph 1). Firstly, tasks T_3, T_5, T_7, T_6 are removed by sequential rule to produce a new reduced graph from (a) to (b). Note that terminal rule does not apply because first (T_1) and last nodes (T_8) are query nodes. Similarly, sequential rule does not apply to T_2 and T_4. The redundant flow between the bottom fork and synchronizer structure (as a result of removing T_6 and T_7 from (a) to (b)) is reduced from (b) to (c) by closed rule. Apply algorithm again, the *fork* node with one outgoing flow is reduced from (c) to (d) by sequential rule, then the upper *synchronizer* node is reduced from (d) to (e) by adjacent rule, and the redundant flow is reduced from (e) to (f) by closed rule. Lastly, apply algorithm again, the lower *fork* node is reduced by sequential rule from (f) to (g). The reduced graph in (g) of process variant V_1 is equivalent to query Q_1.

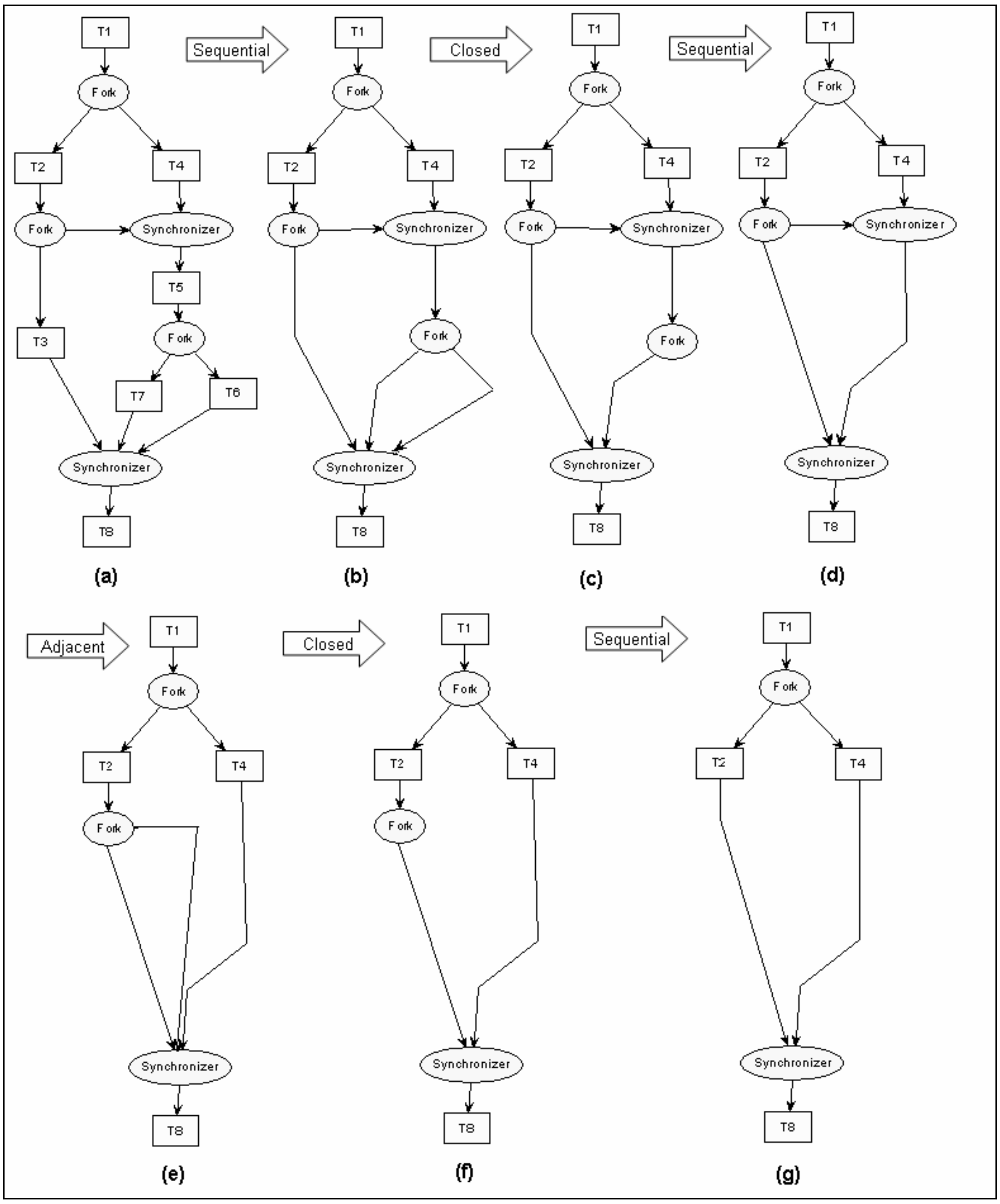

Fig. 6. Applying Selected Reduction Rules on variant V_1

Similarly, applying `SELECTED_REDUCE` on all variants given in Fig.3 for both query 1 and 2, gives reduced structures as illustrated in Fig.7.

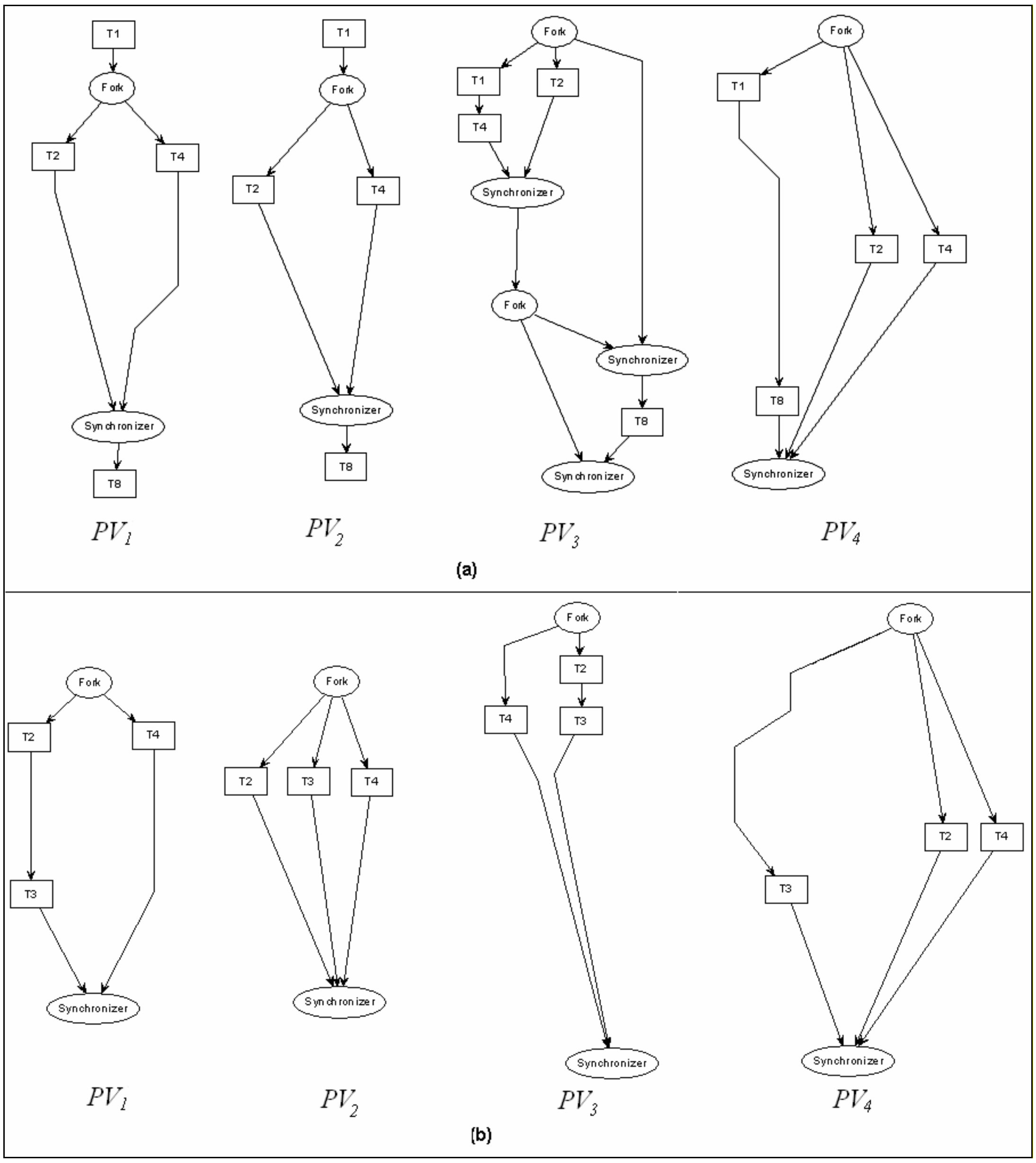

Fig. 7. Results of SELECTIVE-REDUCE on V_1, V_2, V_3 and V_4 against query (a) Q_1 and (b)Q_2

V_1 and V_2 are said to be **exact matches** with Q_1 since the reduced process graph of V_1 and V_2 are isomorphic to query graph Q_1 (as shown in Fig.3). V_3 and V_4 are termed **partial matches** with Q_1 in terms of containing the same set of tasks *T1*, *T2*, *T4* and *T8*, but the process graphs are structurally different from Q_1. Similarly, V_2 and V_4 are exact matches with Q_2 and V_1 and V_3 are partial matches with Q_2.

3.3 Result Interpretation

The result of a query issued to PVR will produce a collection of variants that match the query criteria. In the previous section, we have provided a method through which so called exact matches can be retrieved effectively. However, as Fig.6 illustrates, there may be several partial matches as well, that is variants where the query node set is found, but there are structural differences. A partial match may be significantly different and deemed irrelevant (e.g., variant V_3 against query Q_1 as shown in Fig.6 (a)) or may be quite similar (e.g. variant V_1 against query Q_2 as shown in Fig.6(b)).

The question is, if PVR query results can be further improved by providing ***ranking*** to the set of partially matched variants. We propose a simple method based on flow (edge) counting to provide further insight into partial matches. The method finds out the similarity between a reduced process graph (of partially matched variant) and a query graph by first comparing the number of matching flows between the two, and the similarity degree is given by the percentage of matching flows among the total number of flows in the reduced process variant.

Algorithm 2 RANK_RESULT

```
Input reduced process graph $P$, query graph $Q$
Output Rank of match

  $count \leftarrow 0$
  for each node $n \in N[P]$, $nodeType[n] = Task$ do
      if $InFlows[n] \in F[Q]$ then
          $count \leftarrow count + 1$
      end if
      if $OutFlows[n] \in F[Q]$ then
          $count \leftarrow count + 1$
      end if
return $100\% * (\text{count} / |F[P]|)$
```

The method traverses the reduced graph of a process variant, where each incoming and outgoing flow of task node is compared with those in the query graph. The counter which holds the number of matching flows is increased if a match is found. Note that the flows which connect a coordinator with another coordinator, (e.g., from a fork to a synchronizer) are precluded since transitions represented by these flows do not impose structural constraint on the process graph. Fig 8 illustrates the process of applying RANK_RESULT to partial matching variants in Fig. 6 against query Q_1 and Q_2. Incoming flow (*InFlows*) and outgoing flow (*OutFlows*) of each task node are numbered accordingly for illustration purpose. The numbers for the matching flows (with query graph) are underlined. For example, in the reduced process graph for V_3 (Fig 8a), flows 2, 4, 5, 6 are matching flows against query graph Q_1 (Fig. 4). Since there are 7 non-redundant flows in V_3, the degree of match of V_3 and Q_1 is 100%*(4/7)=57.1%. As the result of RANK_RESULT, V_3 is ranked at 57.1% for Q_1 and 80% for Q_2; V_4 is ranked at 66.7% for Q_1; V_1 is ranked at 80% for Q_2.

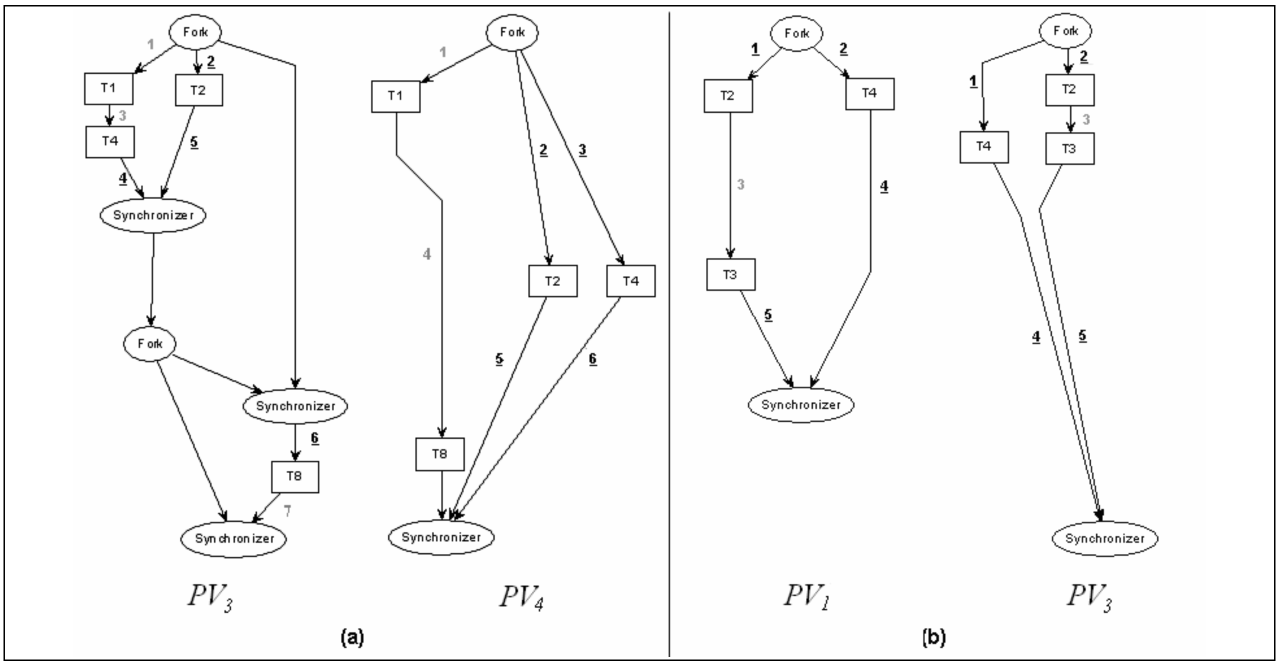

Fig. 8. Applying RANK_RESULT on reduced variants V_3, V_4 against Q_1 (a), and on V_1, V_3 against Q_2 (b)

Table 1 presents the summary of query results. Applying SELECTIVE_REDUCE to process variants against the query graph results the reduced process graphs, from which exact or partial match between process variant and query graph can be determined by comparing the structure of the graphs. Subsequently applying RANK_RESULT to partially matched process variants, the degrees of similarity of process variants to query graph can be determined.

Table 1. Summary of Results for Queries Q_1 and Q_2

	V_1	V_2	V_3	V_4
Q_1	Exact Match (Subsume)	Exact Match (Subsume)	Partial Match (Rank: 57.1%)	Partial Match (Rank: 66.7%)
Q_2	Partial Match (Rank: 80%)	Exact Match (Subsume)	Partial Match (Rank: 80%)	Exact Match (Subsume)

3.4 Query Refinement

As in any query system, the ***preferences*** of a query will be determined by the query designer. For example, the user may set a preference to retrieve on exact matches, or may set a preference to retrieve partial matches against a given threshold, e.g. ≥80% match. Other structural matching techniques for graphs [11, 12] can also be applied if precise ranking of partial matching variants is required.

Lastly, the query facility of PVR can potentially retrieve a very large result, which may not be the best value for utilizing PVR as an effective information resource. It will be important to provide functionality to further refine query criteria. ***Multi-aspect queries*** will play an important role, where structural search is combined with search on operational properties, e.g. find process variants that correspond to the structure of query 1 (Fig.4) AND no test was performed by a senior engineer.

4 Related Work

Process models have been regarded as an information resource in many aspects of modern enterprises. [13] points out that process models, which contain constraints, procedures and heuristics of cooperate knowledge, are regarded as assets of enterprises. A process model may serve as a template for a class of similar business processes performed within an enterprise. The collection of process models are often regarded to as the knowledge base for enterprise operations. The modeling and enactment of process models have been an intensive theme of workflow research in last decade.

However, in recent years, with the growing interest in BPM, process diagnosis and redesign activities of process lifecycle are being targeted [14, 15]. The diagnosis activity referred to as Business Process Analysis (BPA), which involves monitoring and analysis of process execution patterns and performance. Workflow mining is one such approach, which is to extract information about processes from transaction logs [16]. The focus of workflow mining is on the techniques to derive meaningful workflow models with as little information as possible for analysis purpose. On the other hand, the redesign activity emphasizes on the reuse of past instances of process execution to achieve new operational goals in similar situations. The difference in focus determines a slightly different approach for process redesign.

Process redesign, or process reengineering, often refer to the so-called “best practices”, or the best way to perform a particular type of process [17]. The measures of “best practices” are manifold. For example, in the resource utilization perspective, a process is the best practice among others if least performers are required. While in other perspectives, the process with largest instances of the same type can be the best as it is most trusted. These measures correspond to business metric [15]. The most common way to capture, maintain, manage and diffuse knowledge associated with the best practices can be found in knowledge-based systems [17].

Knowledge-based systems have been used in diverse contexts for BPM, a typical example is to build a repository of past processes to provide reference for diagnosing and resolving runtime exceptions [18].

Another predominant technique has been Case-Based Reasoning (CBR) based workflow management [19 - 23]. CBR is a problem-solving paradigm from the Artificial Intelligence (AI) community, which utilizes specific knowledge of previously experienced problems and solutions to solve new problems in similar situations. CBR techniques have been applied to guide schema redesign and dynamic schema changes, as well as to provide learning facility for business process evolution. These approaches have demonstrated the possibilities of utilizing CBR techniques to achieve workflow management goals. However, there are still some shortcomings in those approaches that can be improved.

Firstly, process models kept in the knowledge base are not domain-specific. For example, in [19], workflow models are collected from best practice handbook [24], which contains a board spectrum of business processes from different business domains. It is questionable whether the collected process models are specific enough to provide references in a particular domain of business application.

Secondly, process models are complex objects, which can be perceived from many different aspects (e.g., control dependencies, resource allocation, time constraints etc.). The representations of process models need to be in a form such that matching and retrieval operations could be effectively performed, i.e., similarity between process models can be precisely defined on the abstractions. Current approaches match process models either according to the similarity of the task names [20], or by the textual descriptions of the models (e.g., In [22], process models in the repository is identified by a set of question-answer pairs.).

Lastly, the requirement of case retrieval (query criteria) needs to be formalized such that systematic query evaluation methodology can be defined.

The approach presented in this paper is motivated by the need to provide support for specification and evaluation of queries on complex structural features. This is further assisted by the use of process variants, all of which provide minimal guarantees of design quality. The proposed process variant repository serves as a knowledge base for process redesign and analysis purposes. The schema of process variants defines an abstraction of process models properties where queries on process variants can be effectively formulated and processed. Algorithms for evaluating structural similarity of process models are presented, which are able to precisely distinguish complete and partial matches of process models.

5 Summary

Variations in work practice often represent the competitive differentiation within enterprise operations. In this paper we have argued for the value of variants in business process management platforms. We utilize a constraint based process modeling and execution framework that allows controlled variance in process instances. By doing so, the quality of the variants can be ensured as each recorded variant enforces essential process constraints, but may still exhibit dramatically different structural and operational properties. This paper provides methods to benefit from a repository of process variants, namely PVR. The presented methods provide effective means of searching and matching process variants against a given query, and generate result sets that can be conveniently ranked. The work reported in this paper focuses on queries on the structural aspect, but can be extended to multi-aspect queries.

The results of the proposed query facility in PVR can provide deep insights into ongoing work practices, identify areas of process improvement, and contribute to systematic and well-informed process evolution.

6 References

1. Ellis, S., Keddara, K., Rozenberg, G.: Dynamic Changes within Workflow Systems. In: Proc. ACM Conference on Organizational Computing Systems (COOCS 95), Milpitas, CA. USA, (1995)
2. van der Aalst, W. M. P., t. Hofstede, A. H. M.: YAWL: Yet Another Workflow Language. Information Systems, Vol. 30, (2005) 245-275
3. Scheer, A.W.: ARIS - Business Process Modeling. 3ed, Springer, Berlin, Heidelberg, New York: (2000)
4. Jablonski, J S., Bussler, C.: Workflow Management - Modeling Concepts, Architecture and Implementation. International Thomson Computer Press (1996)
5. Delphi Group.: BPM 2005 Market Milestone Report. (2005) URL: http://www.delphigroup.com/research/whitepapers.htm
6. Lu, R., Sadiq, S., Padmanabhan, V., Governatori, G.: Using a Temporal Constraint Network for increased flexibility in Business Process Execution. In: Proc. Seventeenth Australasian Database Conference (ADC2006), Hobart, Australia (2006)
7. Sadiq, S., Sadiq, W., Orlowska, M.: A Framework for Constraint Specification and Validation in Flexible Workflows. Information Systems, Vol.30(5), Elsevier Science (2005) 349 - 378
8. Aamodt, A., Plaza, E.: Case-based reasoning: foundational issues, methodological variations, and system approaches. AI Communications, **7**(1) (1994) 39--59
9. Sadiq, W., Orlowska, M.: Analyzing Process Models using Graph Reduction Techniques. Information Systems, 25(2) (2000) 117 - 134
10. Sadiq, W.: On Verification Issues in Conceptual Modeling of Workflow Processes. PhD Thesis, The University of Queensland (2002)
11. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In: Proc. 18th International Conference on Data Engineering. San Jose, USA (2002)
12. Rahm, E., Bernstein, A.: A Survey of Approaches to Automatic Schema Matching. VLDB Journal, Vol.10 (2001) 334 – 350
13. Leymann, F., Altenhuber, W.: Managing Business Processes as an Information Resource. IBM Systems Journal, 33(2) (1994)
14. van der Aalst, W. M. P., t. Hofstede, A. H. M., Weske, M.: Business Process Management: A Survey. In: Proc. Business Process Management: International Conference (BPM 2003), Eindhoven, The Netherlands (2003)
15. Casati, F.: Industry Trends in Business Process Management: Getting Ready for Prime Time. In: Proc. 16th International Workshop on Database and Expert Systems Applications (DEXA 2005), Copenhagen, Denmark (2005)

16. 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
17. O'Leary, D., Selfridge, P.: Knowledge Management for Best Practices. Commun. ACM, Vol.43(11) (2000)
18. Dellarocas, C., Klein, M.: Integrating Process Learning and Process Evolution - A Semantics Based Approach. In: Proc. Business Process Management, 3rd International Conference (BPM 2005), Nancy, France (2005)
19. Madhusudan, T., Zhao, L.: A Case-Based Framework for Workflow Model Management. In: Proc. Business Process Management: International Conference (BPM 2003) Eindhoven, The Netherlands (2003)
20. Madhusudan, T., Zhao, L., Marshall, B.: A Case-Based Reasoning Framework for Workflow Model Management. Data Knowledge Engineering, Vol.50(1) (2004) 87-115
21. Rinderle, S., Weber, B., Reichert, M., Wild, W.: Integrating Process Learning and Process Evolution - A Semantics Based Approach. In: Proc. Business Process Management, 3rd International Conference (BPM 2005), Nancy, France (2005)
22. Weber, B., Rinderle, S., Wild, W., Reichert, M.: CCBR-Driven Business Process Evolution. In: Proc. 6th International Conference on Case-Based Reasoning (ICCBR 2005), Chicago, USA (2005)
23. Kim, J.H., Suh W., Lee, H.: Document-Based Workflow Modeling: A Case-Based Reasoning Approach. Expert Systems with Applications, Vol.23(2) (2002) 77 - 93
24. Malone, T. W., Crowston, K., Lee, J., Pentland, B., Dellarocas, C., Wyner, G., Quimby, J., Osborne, C., Bernstein, A., Herman, G., Klein, M., O’Donnell. E.: Tools for Inventing Organizations: Toward a Handbook of Organizational Processes. In: Proc. 2nd IEEE Workshop on Enabling Tech. Infrastructure for Collaborative Enterprises (1993)