Utilizing Successful Work Practice for Business Process Evolution[1]

Ruopeng Lu, Shazia Sadiq, Guido Governatori
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, Australia
{ruopeng, shazia, guido}@itee.uq.edu.au

Abstract

Business process management (BPM) has emerged as a dominant technology in current enterprise systems and business solutions. However, business processes are always evolving in current dynamic business environments where requirements and goals are constantly changing. Whereas literature reports on the importance of domain experts in process modelling and adaptations, current solutions have not addressed this issue effectively. In this paper, we present a framework that utilizes successful work practice to support business process evolution. The framework on one hand provides the ability to use domain expert knowledge and experience to tailor individual process instances according to case specific requirements; and on the other, provides a means of using this knowledge through learning techniques to guide subsequent process changes.

1. Introduction

In modern enterprise information systems, workflow technology has been extensively used to automate the coordination of business processes. Workflows represent the organizational flow of control and information from one processing entity to another [1,2]. Traditionally, workflow systems are designed to automate business processes that contain repetitive procedures and can be modelled prior to deployment. New requirements for workflow modelling have risen due to the deployment of workflow systems in non-traditional domains such as collaborative applications and highly dynamic processes that demand certain level of flexibility in execution. It is essential that technology supports the business to adapt to changing conditions, where different process models could be derived from existing ones to tailor individual process instances.

[1] This work is partly supported by the Australian Research Council funded Project DP0558854

An example can be found in customer inquiry management of a telecommunication company, where inquiry logging and reporting procedures are predictable and repetitive. However, response to individual inquiries and subsequent tests and services performed are prescribed uniquely for each case, but nonetheless have to be coordinated and controlled. Suppose that a number of diagnostic tests, (say 5 tests, *T1, T2, ...T5*), are available. Any number of these tests can be prescribed for a given request, and in a given order. The supervising engineer should have 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, e.g., supervising engineer in this example. Most often such decisions can only be made based on case specific conditions that cannot be anticipated at design time.

The problem lies with the fact that some of the requisite knowledge 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]. With the absence of explicit articulation, the complexity is increased manifold.

Furthermore, industry studies [4] show that 53% 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) instance specific conditions.

The aim of this paper is to provide a foundation for designing and implementing a framework for harnessing successful work practice driven by domain experts, and subsequently providing a means of using this knowledge for business process design and evolution.

The rest of the paper is organised as follows. Section 2 will present the building blocks of the modelling and execution framework. Section 3 will discuss related work. In section 4, details of approach will be presented. Conclusions drawn from this work will be presented in section 5.

2. Framework for Business Process Modelling and Execution

Where as traditional workflow technology has primarily targeted automated flow of control, we have argued in the above discussion that a degree of flexibility is critical to cater for dynamic business environments. Providing a workable balance between flexibility and control is indeed a challenge, especially if generic solutions are to be offered. Clearly, there are parts of the process which need to be strictly controlled through fully predefined models. There can also be

parts of the same process for which some level of flexibility must be offered, often because the process cannot be fully predefined due to lack of data at process design time.

We propose a framework for business process modelling and execution that attempts to achieve a balance between flexibility and control [5]. The framework consists of two major components: (1) A constraint-based process modelling approach, called Business Process Constraint Network (BPCN); and (2) a repository for case specific process models, called instance template repository (ITR). BPCN provides a platform for dynamic process modelling and execution. While ITR provides a well-formed structure to store past process designs, as well as an instrument to harness successful work practice for process evolution. These are introduced below.

2.1. Basic Terminology

We consider a business process as consisting of a set of tasks, where a task is either an atomic activity or a sub-process that contains one or more activities. The process of extracting relevant facts from the business process logic and representing them in a process model with language-specific syntax and semantics is called process modelling. Typically, the process model is then verified and validated in terms of semantic completeness and structural correctness according to certain criteria [6]. The process model is then executed in a runtime environment, where the process engine creates multiple process instances, coordinates execution of tasks and handles runtime exceptions based on the process model.

A process graph (figure 1) presents a typical graphical process model, where coordinator nodes (ellipses) represent typical semantics [7]. We will be using this notation to illustrate various examples in this paper.

A process instance is a particular occurrence of a business process. In our framework, the notion of a process instance is two-fold, which we explain in the next section.

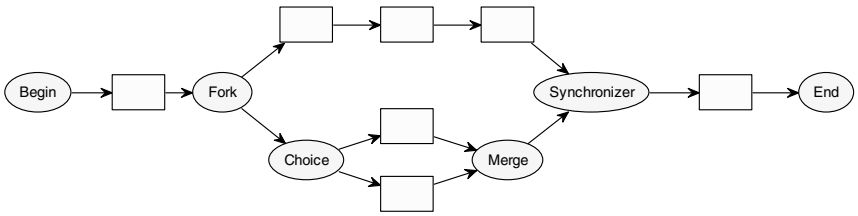

Figure 1. Graphical modelling structure.

2.2. Framework Overview

In general, the requirements of process models are described by multiple aspects [8] such as control flow, data flow, resource allocation and temporal properties

etc. Various aspects are intended to express the **constraints** under which the business process can be executed such that the targeted business goals can be effectively met.

We believe that there are certain types of processes, which would greatly benefit from a modelling approach, in which a minimal number of essential requirements (constraints) are captured in the process model. For example, in the scenario of customer inquiry management (as discussed in section 1), a process model can be specified (figure 2). RE, AS and LR represent the tasks of receive inquiry, assess situation and log report respectively. Note that a part of the process remains flexible. It contains a set of unstructured tasks (tests T1 to T5), however, the exact execution order is determined by some domain experts at runtime according to needs of specific cases. Trying to predict all possible configurations in such a case would either be simply impossible, or at least not be conducive to efficient (customized) response.

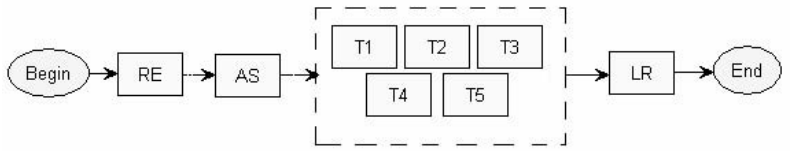

Figure 2. Process model for customer inquiry management.

As long as instances conform to the essential process requirements, they are considered valid. In other words, instance-specific models can be designed at run time, by domain experts, by utilizing their experience and freedom of practice as long as the instance template does not violate the given constraints. The basic premise is that for a class of business processes it is possible to specify a small number of constraints, but allow for a large number of execution possibilities at runtime.

Furthermore, although satisfying the same constraints, instances designed by domain experts may vary significantly. Over time, the repository of such instance templates 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.

The business process (or a part of the process) is modelled through the BPCN. The result of the design and execution is retained in ITR in order to concretise implicit process knowledge and generalise design preferences. The following diagram (figure 3) illustrates the overall architecture of the framework.

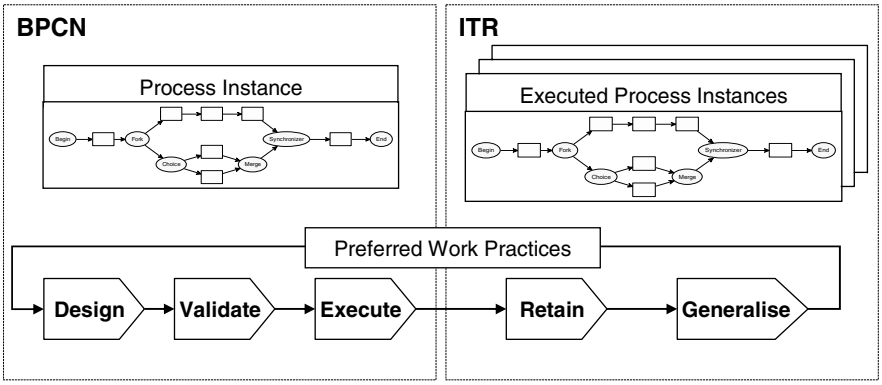

Figure 3. Framework overview.

2.3. Business Process Constraint Network

BPCN has been developed to formalise the problem of flexible business process design [9]. The rationale of the BPCN approach is to provide a *descriptive* way to build models for business processes where complete process cannot be prescribed at design time. BPCN captures the tasks in a business process, their properties, and the process constraints applicable to the business process. In BPCN, business process semantics is extracted and transformed into a set of process constraints served as process requirements that must hold for all process instances. Such constraints are minimal and conflict-free, which are represented in a way that is readable by human and supports analysis and validation for correctness. BPCN uses constraint satisfaction techniques to model process constraints, as well as to determine constraint network consistency. The process model, instead of being explicitly prescribed at design time, is constructed at runtime and validated against the process constraints.

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. We refer to these individually tailored process instances as **instance templates**. This conformance is guaranteed through constraint validation algorithms [9]. The execution environment allows the generation of potentially a large number of customized instance templates, each of which has been constructed with the help of a domain expert utilizing expert knowledge as well as case-specific requirements.

Figure 4 presents the overall BPCN approach for business process modelling and execution. The main processes of BPCN are as follows:

1. The process model is defined including the BPCN specification.
2. The process model is verified and the BPCN is checked for consistency.

3. The definition created above is uploaded to the process engine for execution. A (default) instance template of the process model is created for execution.
4. 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.
5. Runtime user (dynamic instance builder) may decide to change the instance template.
6. Instance template must be verified and validated against the constraints of BPCN.
7. Newly defined (or revised) instance template resumes execution.

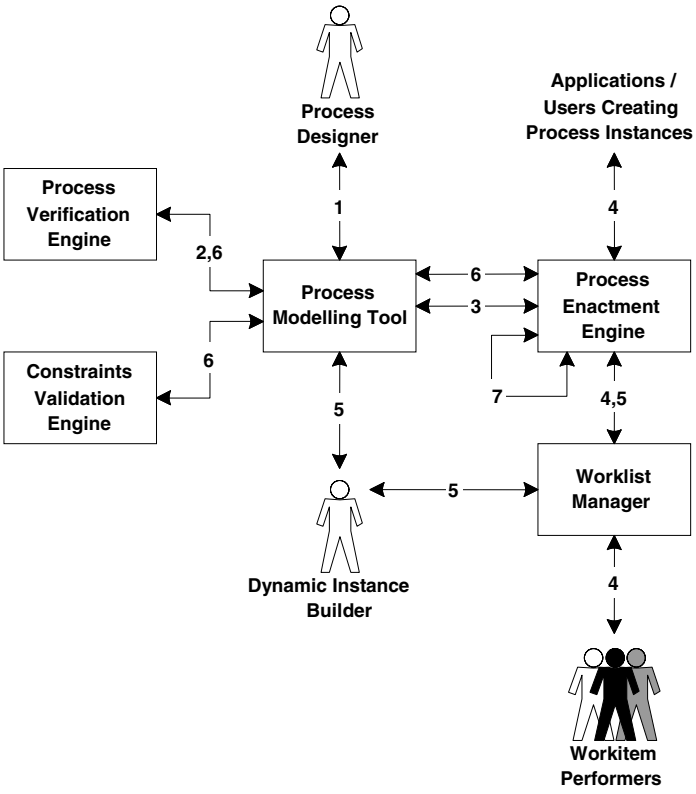

Figure 4. Summary of the BPCN approach.

2.4. Instance Templates Repository

The ITR is a structure for storing instance templates. A **case** is referred to as an instance template stored in the instance template repository. The instance template repository can also be also called the **case base**. The main contribution

of this paper is the management of the ITR, specifically its role in utilizing successful work practice for business process evolution.

It is important to note that the way that domain experts reason about the situation during process modelling cannot be truly reconstructed using computational techniques. However, the purpose of the ITR is to somehow extract the knowledge and reasoning that led to a particular design. We argue that these design decisions are embedded in various process properties e.g., process data values. If a significant number of instance templates have been designed in a similar way, then this is an indication of a preferred (and likely successful) work practice. However, this raises a number of questions relating to process properties, similarity measures, template frequencies, and the need for an overall technology that effectively provides these functions for ITR management. These questions are briefly introduced below.

The first question relates to the **schema** design of instance templates, that is what process attributes need to be stored and how in order to assist future reasoning. Secondly, there is the issue of the **indexing** structure of ITR, which defines how the templates are organized such that update and retrieval operations can be effectively performed. Thirdly, we need a **similarity measure** of two instance templates that provides evidence for cases matching. Finally, the **update** algorithm specifying the process of populating ITR, as well as generalising successful work practices.

We collectively refer to the design of the ITR, the approach to update it, as well as the techniques devised for generalising successful work practices from it as the ITR framework.

The answers to the above questions constitute the design goals of our instance template repository and will be addressed in section 4 where the detail approach of ITR is presented. The solutions to some of these questions are analogous to another problem-solving approach, which is Case-Based Reasoning (CBR). Accordingly, in section 3 we will discuss the background of case-based reasoning, as some aspects of the ITR framework are based on CBR. Some workflow modelling approaches based on case-based reasoning will also be presented.

3. Related Work

The difficulty in mapping of domain specific knowledge to machine readable representations is widely recognized in literature. This is a hard problem in general and has been addressed in process modelling through various approaches, e.g., through modular approaches [8] where process logic is divided into different perspectives (informational, functional, organisational etc.); through language expressibility analysis in workflows [10]; and through comprehensive methodologies [3].

In this paper, we approach this problem through user's perspective, by providing a mechanism to push successful work practice preferences into process

specification. This problem actually translates to finding the "best" practice for a given scenario.

There are several techniques that address similar problems of knowledge acquisition and learning. Case-Based Reasoning (**CBR**) is one such approach, that we have found to link closely to our problem. CBR is a problem-solving paradigm from the Artificial Intelligence (AI) community, which utilises specific knowledge of previously experienced problems and solutions. By definition, a case is a piece of contextualised knowledge fundamental to the reasoning goals [11]. A new problem is solved by adapting similar past cases to meet new demands. Moreover, CBR also supports incremental, sustained learning, as new experience is retained when a new problem has been solved, which provides references for future problems. CBR has been proven to be useful in weak theory domains and system with a large number unstructured and experiential knowledge [12,13]. Interested readers can refer to [11-13] form a formal treatment of CBR.

Our approach applies certain concepts from case based reasoning by following the general design guidelines of case representation and repository (section 4.1), as well as adapting the indexing and case matching techniques (section 4.1– 4.3).

There are a number of recent proposals for workflow modelling approaches based on case-based reasoning techniques [14-17], which have demonstrated the possibilities to utilise CBR techniques to achieve workflow modelling goals. However, there are still many shortcomings in those approaches that needed to be improved. Firstly, the solution proposed by CBR is only as good as the cases it collects, since the past cases are the only knowledge a CBR system possesses. Thus, workflow models kept in the system needed to be sensible and domain specific. Secondly, because index is important, especially for efficient case retrieval and update in large case bases, a proper organisation of case memory is also highly desirable in CBR-based workflow systems. Lastly, a precise definition of similarity is needed if effective case matching is required.

4. Approach

In this section, we present the detail approach to design and implement of the instance template repository.

4.1. Schema for Instance Templates

When a process instance completes execution, the template that governs the instance (as designed through BPCN) is stored in the ITR. The **schema** of instance template repository defines the structure and data content according to which instance templates are stored.

Ideally, the schema should include all descriptive information taken into account during template design [11]. The fundamental goal of ITR is to extract

preferred work practices, more specifically to generalise the conditions contributing to the preference. The descriptive information in instance templates are the properties of business processes. These properties can be divided into two levels, instance level and task level. In our approach, we classify instance level features into **temporal** aspect (process execution duration), **control flow** aspect (ordering dependencies between process tasks e.g., tasks executed in parallel or in sequence etc), and **data** aspect (values of workflow relevant data[2]). Task level features include **resource** aspect (resource allocated to certain tasks).

We now give the formal definition of the instance template schema.

An instance template I is defined by a tuple<*Id, G, R, D, T*>, where

- *Id* is the identifier of instance template I (i.e., names),
- $G = <N, F>$ is a direct acyclic graph (DAG), where N is the set of nodes and F is the set of arcs (edges) $F \subseteq N \times N$, where the arcs correspond to flow relations. The set of node is partitioned into a set of task nodes and a set of coordinators.
- $R = \{R_1,...,R_k\}$ is a finite set of resource instances allocated to I.
- $D = \{D_1,...,D_l\}$ is a finite set of workflow-relevant data items related to I.
- $T = \{t_1,...,t_n\}$ 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 the 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 $dur_i = |t_i^+ - t_i^-|$.

Duration of an instance template I is the sum of execution durations of all its tasks. i.e., $Duration(I) = \sum_{i=1}^{n} dur_i$.

The instance template repository ITR is the set of all instance templates, ITR= $\{I_1,\ldots,I_m\}$.

The above definition can be illustrated by the following example. In the following figure, we provide three instance templates I_1, I_2 *and* I_3. The templates are intended to represent particular test setups in the customer inquiry management scenario as introduced in Figure 2.

---

[2] Workflow relevant data is used by a Workflow Management System (WfMS) to determine the state transitions of a workflow instance, which may be made available to a subsequent activity or another process instance and thus may affect the choice of the next activity to be chosen [1].

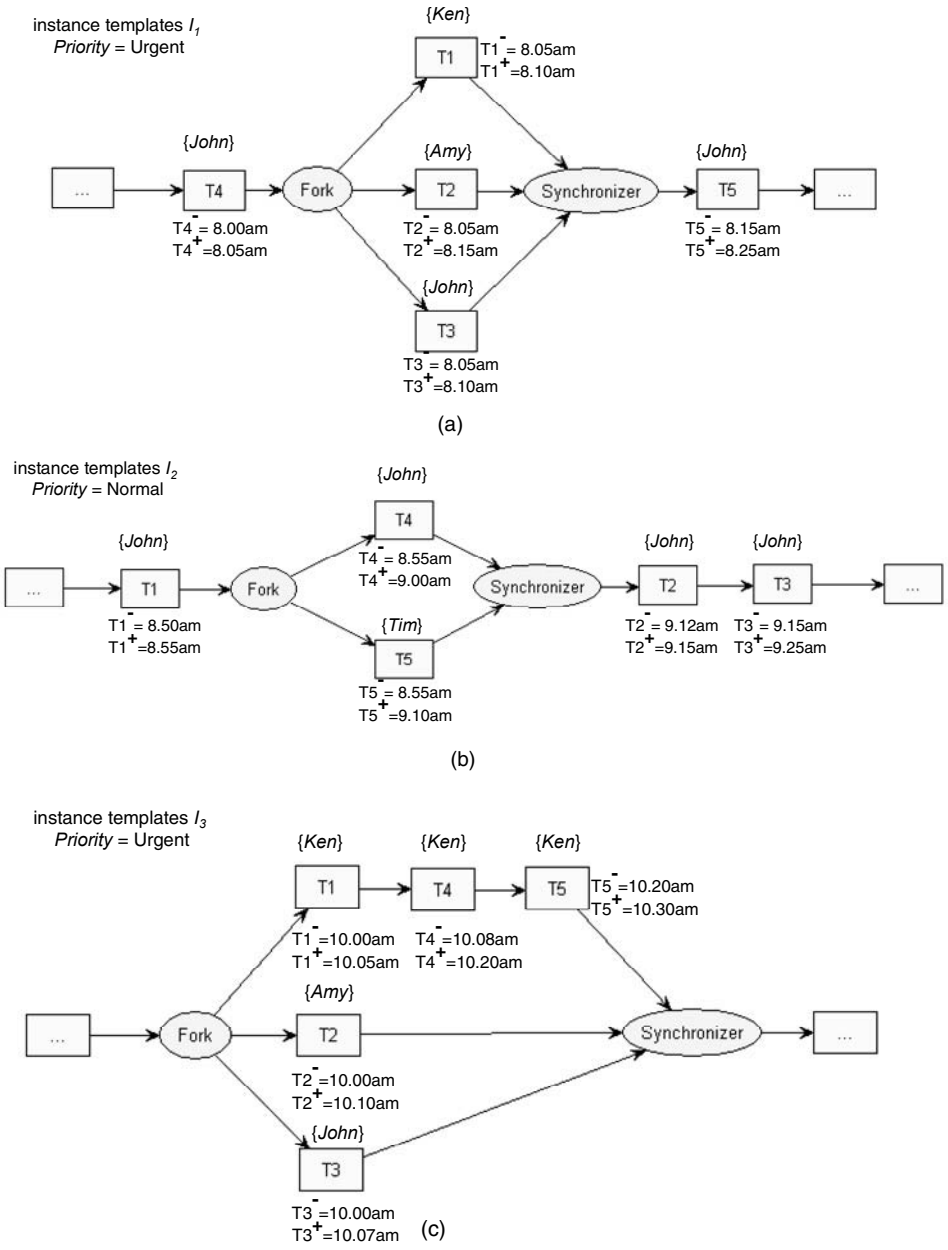

Figure 5. (Part of) Execution graphs of instance template (a) I_1, (b) I_2 and (c) I_3.

I_1, I_2 and I_3 are stored in ITR according to the schema definition, as shown in table 1 and 2.

Table 1. Instance level schema of instance templates.

Id	T (asks)	G(raph)	R(esources)	D(ata)	Duration
1	{*T1,T2,T3,T4,T5*}	<*N,F*>	{John, Ken, Amy}	{Priority = *urgent* }	25min
2	{*T1,T2,T3,T4,T5*}	<*N,F*>	{John, Tim}	{Priority = *normal*}	35min
3	{*T1,T2,T3,T4,T5*}	<*N,F*>	{John, Ken, Amy}	{Priority = *urgent* }	30min

Table 2. Task level schema of instance templates.

Id	n	t^-	t^+	r
1	T4	8.00am	8.05am	John
1	T1	8.05am	8.10am	Ken
1	T2	8.05am	8.15am	Amy
1	T3	8.05am	8.12am	John
1	T5	8.15am	8.25am	John
2	T1	8.50am	8.55am	John
2	T4	8.55am	9.00am	John
2	T5	8.55am	9.10am	Tim
2	T2	9.12am	9.15am	John
2	T3	9.15am	9.25am	John
3	T1	10.00am	10.05am	Ken
3	T2	10.00am	10.10am	Amy
3	T3	10.00am	10.07am	John
3	T4	10.08am	10.20am	Ken
3	T5	10.20am	10.30am	Ken

4.2. Indexing the Instance Templates

While having a populated instance template repository is the first step towards capturing the preferred/successful practices, a major issue is the subsequent retrieval of such practices. How can we identify precedents of similar instance templates against a new one? How can we retrieve them if similar situations arise such that this past experience can be utilised? As the number of instance templates in ITR can be potentially very large, a linear scan for all instances in search for a few "similar" cases is not preferable. These issues contribute to the **indexing problem**. An index essentially serves as identifiers for instance templates sharing some common features, e.g., have been allocated the same set of resources.

We first define some essential terms that we will refer to. These terms follow the general convention as given in [11]. A **descriptor** is an *attribute-value* pair used in the description of an instance template. Descriptors can be of any properties or combination of properties of instance templates. For example, a simple descriptor for an instance template is (*Duration, 25min*). **Dimension** refers to the attribute part of a descriptor. The dimension of the abovementioned example is the temporal perspective of a business process since duration refers to the period of time taken to execute the process instance. Instance templates can be one or more dimensions. A particular descriptor for a given case is also said to be a case **feature**.

In our approach, a hierarchical organisation is used to index cases. A case is indexed by features from multiple dimensions. The collection of all features is called the **description** of a case. The index structure in our approach contains descriptors in four dimensions, namely control flow perspective, resource perspective, data perspective and performance metric perspective.

Control Flow (structural) patterns define the way a workflow management system would order and schedule workflow tasks and it is the primary aspect of a process model, which builds a foundation to capture other aspects of workflow requirements [7]. The execution graph describes the control flow patterns of an instance template, which discriminates instance templates by the topological properties of their execution graph. E.g., (in figure 6) instance templates I_1 and I_2 can be partially distinguished by the control flow pattern of task {*T4, T5*}. Descriptor value {*T4, T5*}-Serial indicates tasks *T4*, *T5* were executed in serial in instance template I_1, while in parallel in I_2. Another control flow feature of I_1 is task *T1*, *T2* and *T3* are executed in parallel.

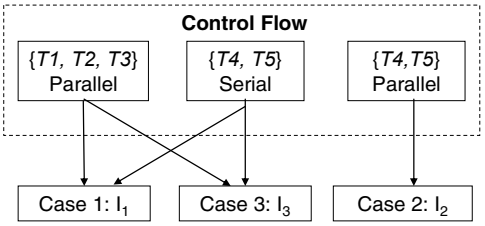

Figure 6. Control flow features.

Resource is another important aspect in business process which generally refers to how resources are represented and utilised in business processes. The resource dimension of ITR index concerns with the resource instances allocated to execute the instance templates. Figure 7 shows the index structure of the resource dimension for I_1 and I_2. This gives an overall resource specification for the templates. Note however that resource aspect may constitute complex specifications, beyond a simple overall list [18].

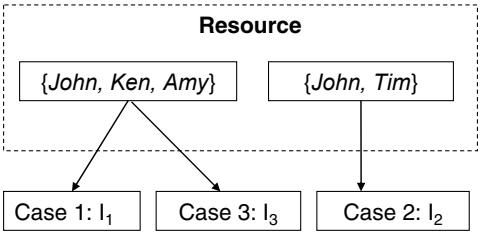

Figure 7. Resource features.

Workflow relevant **data** which may affect design decisions for instance templates can be used as data dimension features in ITR index. Figure 8 shows the data index structure for I_1 and I_2.

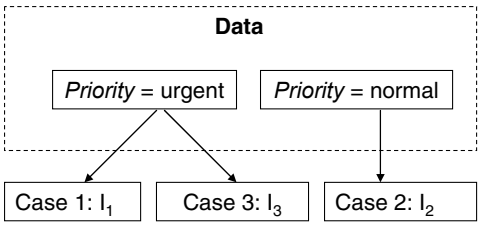

Figure 8. Data features.

The **performance metric** dimension uses quantitative scales to discriminate instance templates based on their execution duration. These may include absolute values, ranges, or temporal relations. Figure 9 shows the index structure for I_1 and I_2.

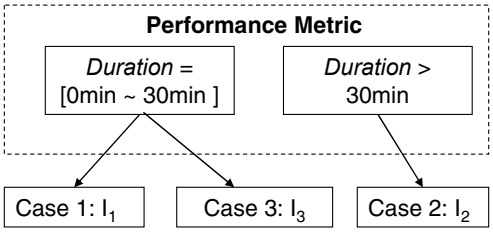

Figure 9. Performance metric features.

The description of a case is the union of all features for this case. For example, the description of I_1, I_2 and I_2 are shown in table 3 and 4:

Table 3. Description for I_1 and I_3.

Descriptor	
Control Flow	{*T1, T2, T3*}-parallel
Control Flow	{*T4, T5*}- serial
Resource	{ *John, Ken, Amy* }
Data	*Priority* = Urgent
Performance Metric	Duration = [0min ~ 30min]

Table 4. Description for I_2.

Descriptor	
Control Flow	{*T4, T5,*}-parallel
Resource	{*John, Tim*}
Data	*Priority* = Normal
Performance Metric	Duration > 30min

Figure 10 shows the overall index structure for I_1 and I_2. The index is a tree-like structure consists of four types of nodes: feature node, description, case node and exemplar node. Feature nodes are in top level which point to one or more descriptions. A description points to a case node or an exemplar node. An exemplar node points to multiple case nodes.

An important concept in the index structure is **exemplar**. A case becomes an exemplar if there are one or more cases having exactly the same description as it does. An exemplar is generalised as a **preferred work practice** when the number of cases in this exemplar exceed the given threshold value (e.g., say 10 cases). Typically, this value is specified by domain experts. This is further described in the section on repository update.

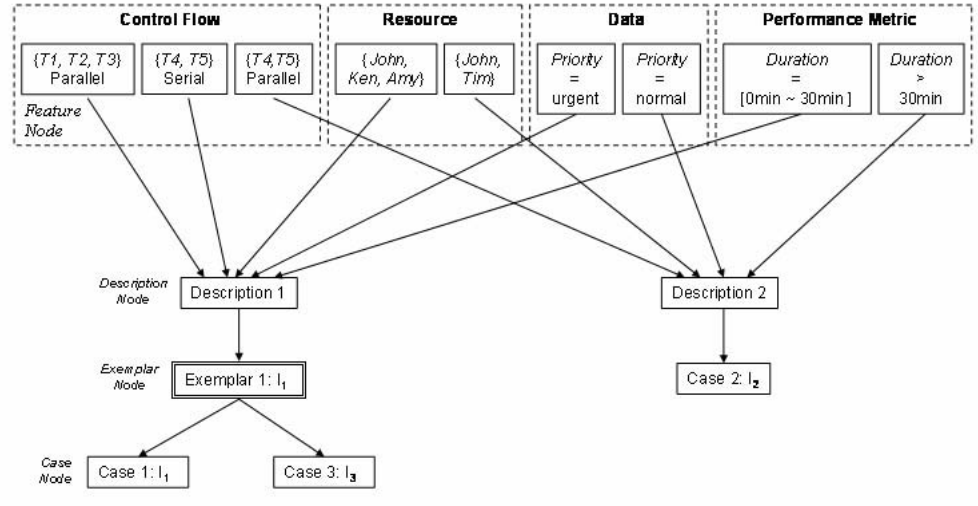

Figure 10. Overall index structure.

The algorithm for inserting new cases into ITR is given in figure11. The *input* to the algorithm is a new case and its description.

```
1.  for each existing description in the index
2.      compare with the description of the new case
3.      if no same description exists
4.          create a new description node
5.          create a new case node under the description node
6.      else  // there exist such description
7.          if case node underneath
8.              Replace case node into exemplar node
9.              Create a new case node under the exemplar node
10.         else  // exemplar node underneath
11.             increase the threshold counter
12.         end if
13.     end if
```

Figure 11. Insertion algorithm.

4.3. Similarity Measures

The ITR is constantly being updated with executed instance templates. We need to determine whether a new instance template *I'* is sufficiently "similar" to the stored instance template *I*. Similar instance templates are stored together and once we have a sufficiently large number of such instance templates, we can generalise them as a preferred work practice. The process of determining similarity between two instance templates *I'* and *I* is called **matching**. To compute the degree of match, two main issues must be considered [11]:

- the degree of match along each dimension, and
- he importance of each dimension

The similarity matching in our approach endorses exact matches along the control, resource and data dimensions. If a numerical value (real numbers between 0 to 1) is assigned to each feature to indicate the degree of match, then 1 indicates exactly matching features and 0 otherwise. That is, the control flow features of two instance templates can be either exactly matching each other (e.g., {*T1, T2, T3*} executed in parallel in both *I* and *I'*), or otherwise not matching at all (e.g., {*T1, T2, T3*} executed in parallel in *I* but sequentially in *I'*). At this stage, we do not consider partial matching within the dimension. Similarly, for resource and data features, two cases match if the set of resource instances (or workflow-relevant data) are the same, otherwise not. Lastly, we measure the degree of match of performance metric features by measuring the distance between two values on a qualitative scale. If two values are within the

same qualitative region (e.g., *Duration* = [1hr ~ 5hr]), then they are considered a match and can be assigned 1, otherwise 0.

We use an aggregate matching function to determine the overall similarity of two cases. For the purpose of illustration, the importance of each dimension is assigned an equal value initially. Let F be the set of all features of the process instance I, we define function Sim: $F \times F \rightarrow \{0,1\}$, which determines the similarity of two features and returns the similarity score 0 or 1. The overall similarity *sim* of instance template I and I' is given by:

$$sim = \frac{1}{m}\sum_{i=1}^{m}(Sim(F_i, F_i^{'}) \bullet W_i)$$

where m is the number of features of I', F_i and $F_i^{'}$ are corresponding features of I and I' respectively, W_i is the weight of feature F_i and $F_i^{'}$. A new case I' is said to be *exactly matching* the base case I if and only if the overall similarity *sim* equals to 1.

For example, supposed a new case I' having the following description (table 5) is to be matched with the base case I_1.

Table 5. Description of base case *I*..

Descriptor	
Control Flow	{*T1, T2, T3*}-parallel
Control Flow	{*T4, T5*}-serial
Resource	{ *John, Ken, Amy* }
Data	*Priority* = Urgent
Performance Metric	Duration > 30min

The two control flow features match each other, thus each having the similarity score 1. Similarly, the resource and data dimensions also match each other respectively, again having the similarity score 1. However, the performance metric dimensions do not match, as duration of I_1 is between [0min ~ 30min], and the duration of I' is > 30min, hence having score 0. The overall similarity is (each dimension having equal weight, i.e., $W_i = 1$ for all i):

$$sim = \frac{1}{5}(1 \bullet 1 + 1 \bullet 1 + 1 \bullet 1 + 1 \bullet 1 + 0 \bullet 1) = 0.8$$

Hence I' is not exactly matching I_1. However, in certain circumstance, when exact matching is not essential or improbable, a threshold value for similarity can be defined (e.g., 0.8) for partially matching cases. In this sense, I_1 and I' *match* each other because the overall similarity score reaches the threshold value 0.8. Furthermore, some dimensions can be prioritised by assigning a higher weight than the others. In order to do so, the weights must be adjusted properly such that the overall similarity score is always between 0 and 1. For example, if in some business process resource feature is more important than performance, the weight for resource features can be given a higher number (e.g., 1.5), while the weight for performance metric is reduced (by 0.5). The overall similarity of I_1 and I' is 0.85, which still qualify as matching cases.

4.4. Repository Update Algorithm

The process of inserting new instance templates and checking against preference threshold constitutes **repository update**. The update algorithm presented below (figure 12) is adapted from [11], which takes the new case as *input*, produces preferred work practice as output, if the number of similar cases in the repository exceeds a predetermined *threshold*.

1. Assess situation, compute index values (description of the new case)
2. Search for matching cases
3. Insert new case in proximity of accessed cases, update index if necessary
4. Check against threshold, if exceeded generalise cases as preferred work practice, update index if necessary

Figure 12. Repository update procedure.

Index selection procedure determines the way the case should be indexed by extracting features from the case and computing the index values. In our approach, the description of the new case is computed by extracting relative information form the instance template according to a preference list of features designed by domain experts, which covering the four dimensions (control, data, resource, performance) of process features. Insertion algorithm (as shown in figure 11) uses the description to insert the case appropriately into the organisational structure of template repository. The organisational structure is reorganized if necessary (when creating a new description, when creating a new exemplar and when an exemplar is generalised as a preferred work practice). When the number of matching cases under an exemplar exceeds the preference threshold as a result of the insertion, the exemplar is generalised as a preferred work practice. The process model (or part of) is updated to reflect the preferred workflow practice.

5. Conclusions and Future Works

In this paper, we have presented an approach for harnessing implicit process knowledge found in successful work practice from a flexible process modelling and execution framework, and using the acquired knowledge for guiding subsequent process design and evolution. The main contribution of this paper relates to the design of functions to manage the instance template repository. The proposed functions are well grounded in known successful techniques in literature. The proposed functions address all aspects of the requirements for ITR management, namely design of the ITR, effective indexing, similarity matching

and consequent repository update. However, we anticipate several interesting and challenging extensions to our work, notably in extending the design of the template schema, and more advanced similarity measure that that allow for partial matching within specific dimensions.

6. References

1. Workflow Management Coalition, *Interface 1: Process Definition Interchange, Process Model*. 1998.
2. W. Sadiq and M.E. Orlowska. *On Correctness Issues in Conceptual Modeling of Workflows*. in *5th European Conference on Information Systems (ECIS `97)*. 1997. Cork, Ireland.
3. A.W. Scheer, *ARIS - Business Process Modeling*. 3ed. 2000, Berlin, Heidelberg, New York: Springer.
4. Delphi Group, *BPM 2005 Market Milestone Report*. (2005) URL: http://www.delphigroup.com/research/whitepapers.htm.
5. S. Sadiq, W. Sadiq, and M.E. Orlowska, *A Framework for Constraint Specification and Validation in Flexible Workflows.* Information Systems, Elsevier Science, 2005. 30(5): p. 349 - 378.
6. W. Sadiq and M.E. Orlowska, *Analyzing Process Models using Graph Reduction Techniques.* Information Systems, 2000. 25(2): p. 117 - 134.
7. W. Sadiq and M.E. Orlowska. *On Capturing Process Requirements of Workflow Based Business Information System*. in *3rd International Conference on Business Information Systems (BIS '99)*. 1999. Poznan, Poland: Springer-Verlag.
8. J S. Jablonski and C. Bussler, *Workflow Management - Modeling Concepts, Architecture and Implementation*. 1996: International Thomson Computer Press.
9. R. Lu, S. Sadiq, V. Padmanabhan and G. Governatori. *Using a Temporal Constraint Network for increased flexibility in Business Process Execution*. in *Seventeenth Australasian Database Conference (ADC2006)*. 2006. Hobart, Australia.
10. van der Aalst, W.M.P., A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. *Workflow Patterns.* Distributed and Parallel Databases, 2003. **14**(1): p. 5-51.
11. J. Kolodner, *Case-Based Reasoning*. 1993: Morgan Kaufmann Publishers.
12. A. Aamodt and E. Plaza, *Case-based reasoning: foundational issues, methodological variations, and system approaches.* AI Communications, 1994. **7**(1): p. 39--59.
13. D.B. Leake ed. *Case-Based Reasoning: Experiences, Lessons & Future Directions*. 1996, AAAI Press/The MIT Press.
14. B. Weber, S. Rinderle, W. Wild and M. Reichert. *CCBR-Driven Business Process Evolution*. in *6th International Conference on Case-Based Reasoning, ICCBR 2005*. 2005. Chicago, IL, USA: Springer.
15. T. Madhusudan, J.L. Zhao, and B. Marshall, *A case-based reasoning framework for workflow model management.* Data Knowledge Engineering, 2004. **50**(1): p. 87-115.
16. J.H. Kim, W. Suh and H. Lee, *Document-based workflow modeling: a case-based reasoning approach.* Expert Systems with Applications, 2002. **23**(2): p. 77 - 93.
17. B. Weber, W. Wild, and R. Breu. *CBRFlow: Enabling Adaptive Workflow Management Through Conversational Case-Based Reasoning*. in *7th European Conference ECCBR 2004*. 2004. Madrid, Spain: Springer-Verlag.

18. N. Russell, A.H.M. ter Hofstede and D. Edmond, *Workflow Resource Patterns*. 2005, Centre for Information Technology Innovation, Queensland University of Technology: Brisbane.