

Process Materialization using Templates and Rules to Design Flexible Process Models

Akhil Kumar¹ and Wen Yao²

¹ Smeal College of Business, Penn State University, University Park, PA 16802, USA

² College of Information Science and Technology, Penn State University,
University Park, PA 16802, USA
akhil@psu.edu, wxy119@psu.edu

Abstract. The main idea in this paper is to show how flexible processes can be designed by combining generic process templates and business rules. We instantiate a process by applying rules to specific case data, and running a materialization algorithm. The customized process instance is then executed in an existing workflow engine. We present an architecture and also give an algorithm for process materialization. The rules are written in a logic-based language like Prolog. Our focus is on capturing deeper process knowledge and achieving a holistic approach to robust process design that encompasses control flow, resources and data, as well as makes it easier to accommodate changes to business policy.

Keywords: flexible processes, rules, templates, materialization, algorithms.

1. Introduction

There are many approaches and frameworks for designing business workflows. Most of them are based on mapping a control flow that specifies the coordination of various activities (see, for instance, [1,2,8,15,21]). The control flow description of a process is also called a *process schema*. In general, there are a large number of process schemas in an organization. This occurs partly because many schemas are variants of one another with minor changes between them. Take for instance, an insurance company that writes policies for automobile, home and other kinds of insurance. When claim applications are made, then the company has to initiate a different process schema for an automobile accident claim as compared to a home damage claim. Moreover, even for a home damage claim, another different process must be enacted for a home whose value is less than \$100,000 versus a home whose value is more than \$250,000 because in the former case only one adjuster might be required to visit the home and appraise the damage, while in the second case two adjusters are required to submit independent reports of damage assessment. In general, if there are thousands of process variants it makes finding the correct process difficult and error prone.

Another complication may arise if the company changes its policy to require two independent assessments only when the value of the home is more than \$500,000. Just this simple business policy change will necessitate a change in many process

schema variants. It is time and effort consuming if every variant of the process schema affected by this change has to be modified. Thus process schema description gets tied into the business policy of the organization.

In this paper we propose a novel solution to process design based on the idea of process materialization. *Process materialization* means to generate on the fly a process schema (say, a BPMN [21] model in XML) that integrates the control flow, resource needs and data from a generic *process template*, which describes a very basic and general process schema, by applying business rules to the input data of a process. In general, these rules correspond to the business policy of an organization. Thus, this is a rule-based approach to process schema design so as to incorporate the business policy in a dynamic way. If policy changes, only the rules have to be modified while the template can remain unchanged. After the schema has been materialized, then the process would be executed by a workflow engine. The main advantage of this approach is that an end user does not have to create a large number of process schemas before hand and manually determine which schema to execute when a case (such as for an insurance claim) arrives. In addition, an end user does not need to modify a large number of process schemas when policy changes occur since the policy can be captured by rules.

Another advantage of this approach is that it leads to *holistic process design*. Workflow research has focused on the modeling of the control flow of a process, while other key aspects like data flow and resource needs of various tasks are neglected. In general, a holistic process model requires additional information like resource needs of each task, data values of parameters associated with a task, equipment and facilities needed for the completion of the task, etc. One of the goals in the proposed approach is to integrate the modeling of resource and data needs of various tasks as well into the process description. Thus, in the insurance example above, it should be possible to specify that: (1) if the damage exceeds \$500,000 then the claim must be approved by a vice-president (a resource related constraint); and (2) at least one adjuster out of two should have more than 10 years of experience (a data related constraint). Modeling approaches that capture such requirements are needed.

Thus, the essence of our approach is: **process template + rules = materialized process**. Naturally, this leads to considerably more flexibility than conventional approaches and is suitable in scenarios involving variability and frequent changes in the environment, as well as resource intensive or ad hoc workflows. The organization of this paper is as follows. Section 2 provides an example to motivate our approach, presents an architecture to formalize this approach and also gives a formal representation for processes, while Section 3 describes our representation approach for rules and discusses rule processing. Then, Section 4 describes our materialization algorithm in detail. Finally, Section 5 provides a detailed discussion and Section 6 concludes the paper with directions for future work.

2. Preliminaries

2.1 Motivating Example

Figure 1 shows an example process template for an insurance process in BPMN notation [21]. In this template, after a claim is received by a customer representative,

it is validated by a clerk to ensure that the customer has a valid policy that relates to this claim. The clerk also makes an assignment to two adjusters who will review and appraise the damage to the auto or the house, and then submit a report. The two adjusters may perform their jobs in parallel. This is indicated by a *parallel gateway* shown as a diamond with a cross. The first parallel gateway where multiple branches split has a corresponding parallel gateway where multiple paths merge. After the reports are received by the customer representative, they are checked for completeness and sent to an officer who will determine the settlement amount based on the reports. Subsequently, two approvals are required by a manager and a senior manager, and then the accounts manager will issue the payment to the customer.

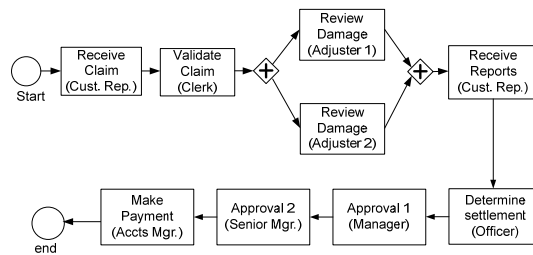


Figure 1: Description of an insurance process in BPMN notation

Figure 1 shows the “normal” tasks, the roles that perform the tasks and the control flow relationships between tasks. However, the realized process may vary depending upon the actual data for a particular incident or case. Thus, the template can be customized for a specific case by applying rules to it. An example rule set is shown in Figure 2. The rules are written in plain English-like syntax. So, if the loss claimed is less than \$500K, then only one adjuster is required (R1); however, if the loss is, more than \$250K then the adjuster should have more than 10 years of experience (R7) and must fill the “long” form (R9). Furthermore, if the loss is more than \$500K, the second adjuster should be classified as an expert (R8). After a settlement is assessed, either one or two approvals are required before payment is made. The number of approvals depends upon the amount of loss (R4, R5, R6). Finally, there are rules related to the urgency status of the case. If it is marked *expedite* then the approvals may be performed in parallel to save time (R2). On the other hand, if it is marked *urgent*, then the second approval may be deferred to after the payment is made (R3).

- R1: If loss < \$500K, **then** skip review by adjuster 2
- R2: If application = expedite **then** perform approvals in parallel
- R3: If application = urgent and loss < \$500K **then**
defer second approval until after payment
- R4: If loss < \$100K, **then** need manager approval
- R5: If \$100K < loss < \$500K, **then** need manager & senior manager approval
- R6: If loss > \$500K, **then** need manager + VP approval
- R7: If loss > \$250K, **then** need adjuster with minimum 10 years experience
- R8: If loss > \$500K, **then** need detailed assessment from an expert
- R9: If loss > \$250K, **then** adjuster must fill the "long" form

Figure 2: Rules to be applied to the process template of Figure 1

Clearly, by applying the rules to the template on different case data, we obtain different processes. Two such materializations for two different cases are shown in Figure 3. In (a), the loss is \$200 K and it is marked *expedite*, while in (b) the loss is \$300K and it is marked *urgent*. We can see that different processes emerge.

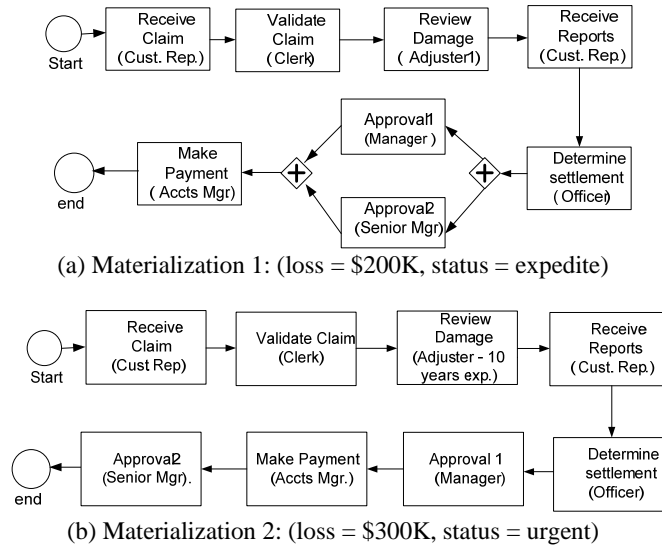


Figure 3: Two materializations from process template and rules

The purpose of this example was to motivate the need for materialization. Our motivation in a nutshell is to reduce the number of processes, especially those which are minor variants of each other, and to make processes more adaptive and agile to business policy changes. Thus, our approach uses process templates to abstract similar processes and uses rules to separate business policy from process design to adjust to a constantly evolving environment.

Next, we give an architecture to show how to formalize this approach.

2.2 Architecture

A high level architecture for our approach is shown in Figure 4. A process designer can create, modify and delete process templates and rules using an editor. Each process template is associated with a number of rules that have the same process id. The editor checks the template for correctness and the rules for consistency. In the case where two rules conflict, the system will give a warning and ask the designer either to modify the rule or associate priorities with them. In addition, the editor checks the executability of rules on their associated template. For example, the editor will give an error warning to the process designer if `delete(task t1)` is contained in the rule while `t1` does not exist in the process template. Detailed definition of semantics for the rule engine is discussed in the next section. As a result, valid process templates

and rules are maintained respectively in the process template repository and shared rule repository. As the business policy changes over time, the process designer can easily change rules associated with a specific process while leaving the process template unchanged. When input data for a particular case is entered into the rule engine by the customer, the rule processing module determines the predicates that are true and passes them on to process materialization algorithm. The materialization algorithm uses these true predicates to modify the process template and create a materialized process instance schema for execution within the process engine. The rule editor can also check for data flow consistencies, i.e. make sure that a task will receive all its input data from the output of previous tasks.

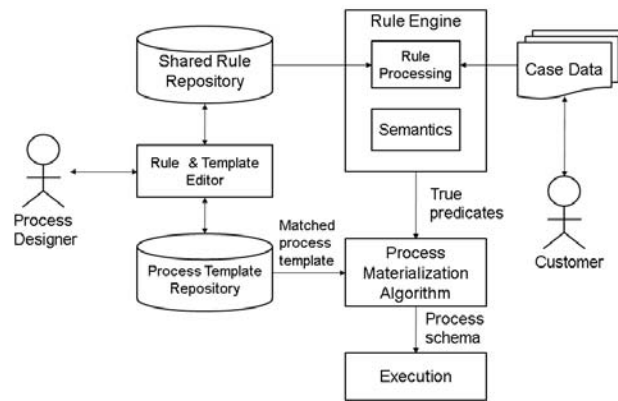


Figure 4: An overall architecture of the materialization approach

2.3 Formal Representation of a Process

A business can be viewed as a collection of processes, and the robustness of these processes to a large extent is a crucial determinant of the success of the business. Business processes can be described using some simple constructs, and most workflow products provide support for these constructs. Four *basic constructs* that are used in designing processes are *sequence*, *parallel*, *decision structure* (or *choice*) and *loop*, as shown in BPMN notation in Figure 5.

In general, business processes can be composed by combining these four basic patterns as building blocks. They can be applied to atomic tasks, e.g. $seq(A,B,...)$ to indicate that tasks A and B (and possibly other tasks) are combined in sequence, or to subprocesses, e.g. $seq(SP1, SP2)$ to indicate that subprocess SP1 and subprocess SP2 are combined in sequence (see Figure 5(a)). Parallelism is introduced by using a parallel gateway to create two or more parallel branches which are synchronized by another parallel gateway as shown in Figure 5(b). We use 'P' or 'Par' to denote the parallel structure. Similarly, a choice structure, denoted as 'C' or Choice, is created with a pair of *exclusive OR (XOR) gateways* and denoted by C or Choice as shown in Figure 5(c). The first XOR node represents a choice or a decision point, where there is one incoming branch and it can activate any one of the two or more outgoing

3 Rule Representation and Processing

3.1 Rule Categories

In general, we are interested in rules related to four aspects of process design:

Control flow rules: These rules may be used to alter the control flow of a process based on input data of a case. In addition to deleting or replacing a task, they can also alter the control flow by moving a task to a different place or changing the relationship of two tasks.

Resource related rules: These rules are concerned with resource assignment based on case data.

Data related rules: These rules are associated with properties or attributes of a resource related to a case or to other case data.

Other rules: These rules may relate to general reasoning, say related to the policies of the organization. An example of such a rule is: "if an insurance claim has not been completed in 7 days, the status is changed to expedite."

These categories can help to organize the rules and define them systematically. However, we show next all these types of rules can be represented in a common way.

<u>Tasks</u> task(t1, receive_claim). task(t2, validate_claim). task(t3, review_damage1). task(t4, review_damage2). task(t5, receive_reports). task(t6, det_settlement). task(t7, approval1). task(t8, approval2). task(t9, payment).	<u>Process structure</u> merge(t1,t2,blk1,seq). merge(t3,t4,blk2,par). merge(blk1,blk2,blk3,seq). merge(blk3,t5,blk4,seq). merge(blk4,t6,blk5,seq). merge(blk5,t7,blk6,seq). merge(blk6,t8,blk7,seq). merge(blk7,t9,blk8,seq).
<u>Roles</u> role(t1,cust_rep) role(t2,clerk) ...	<u>Data</u> data_in(t1,loss) data_in(t1,policy_num) ...

Figure 6: Describing the insurance claim process using predicates

3.2 Rule representation

In Figure 2 the rules were represented informally. They can be written formally in a first order logic language like Prolog [6]. The rules are based on the predicates in Table 1. To illustrate, Figure 7 shows how the rules of Figure 2 will be expressed formally using the predicates from Table 1. The rules are based on case data. Case data is also expressed using the data predicate as follows:

```
data(loss, 10000).  
data(date, 1-Jun-2009).
```

```
data(policy_num, sp34-098-765).
```

Rules R1, R2 and R3 are control flow related rules. R1 deletes task t4 from the process instance if the amount of loss is less than \$500K since the second adjuster review is not required. R2 changes the relationship between tasks t7 and t8 from sequence to parallel if it is marked *expedite*. Similarly, R3 moves the second approval task (t8) to the end of the process if the process is marked *urgent*. Rules R4, R5 and R6 are resource related rules. They are all similar in that they make assignments of resources to tasks based on case data. Thus, in our example, the amount of loss determines the resource that is required to perform a task, such as an appraisal or an approval. R6 requires that if the loss amount is more than \$500K, then a vice-president should perform the second approval task (t8). Finally, Rules R7, R8 and R9 are related to case data such as properties of the resources, or other data.

```
control flow related
R1: delete(t4) :- data(loss, X), X < 500000.
R2: change(t7,t8,P) :- status(proc_id, expedite).
R3: move(t8, Sa, t9) :- status(proc_id, urgent),data(loss,X), X<500000.

Resource related
R4: role(t7, manager) :- data(loss, X), X > 0.
R4':delete(t8) :- data(loss, X), X < 100000.
R5: role(t8, senior_manager) :- data(loss, X),
                                X > 100000, X < 500000.
R6: role(t8, vice_president) :- data(loss, X), X > 500000.

Data related
R7: prop(adjuster, min_exp, 10) :-
    data (loss, X), X > 250000.
R8: prop(adjuster2, qualification, expert):-
    data(loss, X), X > 500000.
R9: prop(form, type, long) :-
    data (loss, X), X > 250000.
```

Figure 7: Different types of rules related to the insurance workflow process template

3.3 Rule Processing and Semantics for Conflict Resolution

When the above rule set is executed on case data, it will apply the predicates that are true to the process template. Assume the case data is as follows:

```
loss = $300k; status = expedite.
```

Now, on adding this data to the rule set we find that the rules that are applicable are: R1, R2, R5, R7 and R9. The corresponding predicates that are true as a result are:

```
Pred1: delete(t4)
Pred2: change(t7,t8,par)
Pred3: role(t8, senior_manager)
Pred4: prop(adjuster, min_exp, 10)
Pred5: prop(form,type,long).
```

These predicates can be applied to the process template in order to instantiate a specific process for this case data. Although all rules for the template are valid and compatible with each other, true predicates generated by them may have conflicting results depending upon the order in which they are applied. For example, **insert**(t2, S_a, t1) and **insert**(t3, S_a, t1) applied to a process consisting of a single task 't1' can result in two processes S(t1, t2, t3) or S(t1, t3, t2) depending upon the order in which these operations are applied. Moreover, sometimes rules may fail. For instance **insert**(t1, S_a, t2) would fail if task 't2' has been deleted by a predicate in the previous step. Therefore, it is very important to specify the following semantics for handling such situations for process materialization. Some possible semantics are:

Arbitrary semantics: do not impose any order on the rules. Assume that all execution orders of rules are satisfactory from the user's point of view and are acceptable. If so, the execution priority of true predicates will follow the order of their corresponding rules in the rule repository.

Priority semantics: assign a priority to rules if the execution order is important. Higher priority rules will execute first, followed by lower priority rules in descending order. The user can assign priority based on the importance of the rules, so the most important predicates will be applied first. Besides, priorities may also be assigned based on timestamps with the more recent rules receiving higher priority.

Fail semantics: return failure when a rule cannot be executed due to different reasons. In this case a process cannot be materialized. Therefore, the user will have to intervene to modify the rules or assign new priorities to them.

It is highly recommended that the user should specify the priority semantics before applying the results of rule processing to materialize a new process. Otherwise, arbitrary semantics will be used automatically. If the materialization process fails, the user will be notified about the predicates that cause the failure.

4 Materialization Algorithm

The rule processing stage generates a list of true predicates that apply to the case data. These predicates relate to control flow, resources, and data of the case. The predicates that relate to the resources (such as role and prop predicates) and also those that relate to data (such as data predicate) are added to the case database directly. For example, prop(form,type,long) assigns the value long to the type attribute of the form object. Such data would be used as input data for task execution. However, the predicates that relate to the control flow are used as input for a materialization algorithm in order to generate a modified control flow schema for the process. The details of the materialization algorithm are discussed next.

4.1 Overview

Our materialization algorithm is based on creating a tree for the generic process template and then applying change operations to it. Each change operation will produce a corresponding operation on the tree. After all the changes are applied to the tree, the resulting tree reflects the control flow of the new process. This can be converted into Prolog rules or into any process description language to describe the process schema. A tree for the process template of Figure 1 is shown in Figure 8. There are several equivalent representations of such a tree. The tasks are at the leaf nodes, while internal nodes are control nodes that give relationships between tasks or blocks of tasks. Accordingly, the internal node labels are prefixed with the *node type* (S, P, C, or L). The child nodes of a sequence node are numbered in order from left to right. Thus, the leftmost child appears first in the sequence and the rightmost one is the last. For parallel and choice nodes, the order of appearance of the child nodes does not matter because of their execution semantics. A loop node has two child nodes, the first one for the forward path, and the second for the reverse, hence the order does matter. This tree can be stored in a tabular data structure as follows: (node_id, type, child node, sequence#).

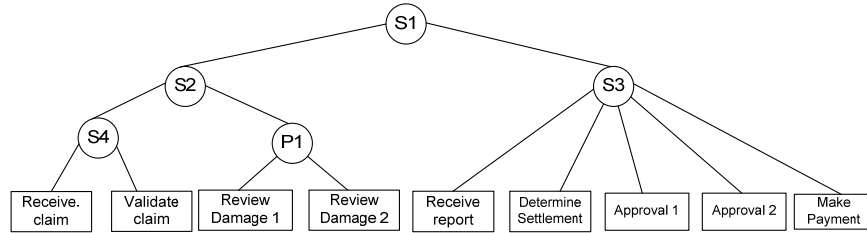


Figure 8: A process tree for the process template of Figure 1

4.2 Algorithmic Details

The control flow modification predicates (or operations) were given in Table 1. They are summarized in Table 2 along with the pseudo-code for implementing them. We discuss these operations next in the context of Figure 8 which is a tree-like representation of the process of Figure 1. Although, this tree only presents the control flow perspective of the process, the resource and data related information can also be included in the nodes. This tree was drawn using the parent child relationships from the base predicates of Figure 6 after applying the rewriting rule 3 (see below). A **delete** operation simply removes the node corresponding to a task in the tree and if the parent of the deleted node has only one child left, then the child is moved up to take the place of the parent. Each non-leaf node should have at least two child nodes and a task is always a leaf node.

When a task is to be **inserted**, its position must be specified in the tree with respect to an already existing task node. Moreover, the relationship between the existing node and a new node should also be specified as sequence (S), parallel (P), choice (C) or loop (L). If it is a sequence it is necessary to state whether the new task is inserted before (S_a) or after (S_b) the current node. The insert procedure is to create a parent

node P1 for the existing node (say, t1) and insert the new node t as a child of P1 in the tree. The parent node can optionally be given a new label N. The **replace** operation simply changes the label of a task node with its new name.

The **move** operation is like a delete followed by an insert. It removes a task from its current location in the tree and inserts it into a new position. This new position is defined with respect to an existing task node in the tree which serves as an anchor node. The **change** operation may be used to modify the relationship between two existing nodes t1 and t2 in the tree. However, this is possible only if a direct relationship (i.e., with a common parent node and no other siblings) exists between the two nodes. Otherwise, the operation would fail. In order to implement this operation, we first check if a direct relationship either exists already or can be found by rewriting the tree into an equivalent tree by means of rewriting rules. If it is possible to do so, then the parent node of t1 and t2 is changed to the new relationship. Otherwise, a failure message is given. The rewriting rules are as follows:

Rewriting rules (A, B, D are nodes of a tree)

1. $\mathbf{P}(\dots, A, \dots, B, \dots) = \mathbf{P}(\dots, B, \dots, A, \dots)$
2. $\mathbf{C}(\dots, A, \dots, B, \dots) = \mathbf{C}(\dots, B, \dots, A, \dots)$
3. $\mathbf{S}(A, B, D, \dots) = \mathbf{S}(\mathbf{S}(A, B), D, \dots) = \mathbf{S}(A, \mathbf{S}(B, D), \dots)$
4. $\mathbf{P}(A, B, D, \dots) = \mathbf{P}(\mathbf{P}(A, B), D, \dots) = \mathbf{P}(A, \mathbf{P}(B, D), \dots) = \mathbf{P}(B, \mathbf{P}(A, D), \dots)$
5. $\mathbf{C}(A, B, D, \dots) = \mathbf{C}(\mathbf{C}(A, B), D, \dots) = \mathbf{C}(A, \mathbf{C}(B, D), \dots) = \mathbf{C}(B, \mathbf{C}(A, D), \dots)$

Nodes A, B, and D could represent either tasks if they are leaf nodes or root nodes of subtrees if they are internal nodes. The first two rules capture commutativity of the parallel and choice operations. The next three rules reflect associativity of sequence, parallel and choice operations. Thus, by Rule #3, three tasks in a sequence (i.e. a parent S with child nodes A, B and D) can be rewritten in a two level deep tree with a parent (say, S1) having child nodes S2 and D. The child node S2 in turn has two child nodes A and B. Clearly, both these structures are equivalent. The same argument applies to parallel and choice nodes.

Continuing with the running example, the result of rule processing (section 3.3) will send control flow predicates pred1 and pred2 to the process materialization algorithm. In the tree of Figure 8, the task Review Damage 1 and Review Damage 2 are executed in parallel. Applying Pred1 will remove Review Damage 2 so the parent node P1 only has one child node. Following the algorithm, we move task Review Damage 1 to replace P1. To apply Pred2, we consider the tasks Approval 1 and Approval 2. They have a sequence relationship between them; however, their parent node S3 has five child nodes. In order to change this relationship between only these two tasks to P, we can use rewriting rule #3 to rewrite the part of the tree under S3 as follows:

$$\mathbf{S3}(\text{rec.}, \text{det.}, \text{app1}, \text{app2}, \text{pay}) = \mathbf{S3}(\mathbf{S5}(\text{rec.}, \text{det}), \mathbf{S6}(\text{app1}, \text{app2}), \text{pay})$$

Now, since app1 and app2 have a common parent S3, we can rewrite as:

$$\mathbf{S3}(\mathbf{S5}(\text{rec.}, \text{det}), \mathbf{S6}(\text{app1}, \text{app2}), \text{pay}) \rightarrow \mathbf{S3}(\mathbf{S5}(\text{rec.}, \text{det}), \mathbf{P2}(\text{app1}, \text{app2}), \text{pay})$$

On the other hand, note that it would not be possible to change the tree such that tasks receive report and make payment are in parallel. On applying the true predicates Pred1 and Pred2 to the process template tree of Figure 8 by using the materialization

5 Discussion and Related Work

Most process design techniques lead to *rigid* processes where policy is "hard-coded" into the process schema thus reducing flexibility. The motivation behind our approach in this paper is to overcome this drawback by integrating rules with generic process templates to materialize processes. The template captures the essence of the flow, while the rules allow modification based on case data, policy changes, resource availability, etc. The new approach is also more holistic since it can go beyond control flow and also capture case data and resource aspects of the workflow.

A preliminary proposal for process and rule integration is given in [18]. Other related work with similar objectives pertains to configurable models [25] and aggregate models [23] although they are not based on rules directly. Direct integration of the process and rule paradigms into a product is being made in the Drools Flow project [13]. Their goals are similar to ours, but they do not create a materialized process. Rather they provide a rule modeling construct or module as a way to include rules into their process. The rules module can be evaluated and a decision can be made accordingly. The advantage of our approach is that it can be integrated into existing process modeling paradigms. On materialization our process models can be expressed in any existing language (e.g., BPMN, BPEL and XPD) and executed in an existing workflow engine. The generic process templates can also be written in any standard language.

There has been considerable amount of related work on execution of dynamic processes in the context of exception handling. The focus there is on modifying a running process when exceptions occur due to failed tasks, erroneous information, etc. However, this body of work is complementary to our work since it focuses on flexibility at run time. In contrast we are more interested in flexibility at design time.

The benefits from design time flexibility we foresee are:

- (1) It leads to a cleaner process design. Thus, if the two materializations shown in Figure 3 were combined into one process it would become very difficult to read.
- (2) It allows separation of organization policy from process flow. The basic template captures the essence of the main process flow by including the tasks and the normal order in which they are performed. However, variations to the generic process flow represent business policy, and these are captured more naturally through rules. When changes in the policy occur, only the rules are modified without affecting the generic process template.

Techniques for supporting dynamic change are discussed in [14,20,22,24] and elsewhere. The focus of this work is to allow operations like task insertion, deletion, etc. to be performed on running workflows in response to exceptions [5,7,12]. There have also been other approaches on designing flexible workflow models: based on deadline based escalation [3], satisfaction of constraints [19] or restrictions [11], availability of resources [17] and on graphs [26]. Another approach for designing workflows is centered on entities [4]. While all these methods try to reduce rigidity of a strict control flow approach, they lack the flexibility of rules. In [20], a rule based approach for dynamic modification in a medical domain is given with the focus on handling of exceptions. Rules can also be used to ensure that business processes comply with internal and external regulations. These rules could be specified in a first-order language [16] or in deontic logic [9].

Of course, on the downside additional cost is involved in managing the rules and ensuring consistency. However, if rules are created through a user friendly interface, then the burden on the user is minimized. Moreover, for the most part, we expect rules to be simple and their interactions few, thus reducing complexity. Another drawback with our approach is that it may give too much freedom to users to make ad hoc changes in business processes. This can be restricted by adding controls in the form of meta-rules that restrict use of modification operations only to certain users.

6 Conclusions

This paper described a novel proposal for designing flexible business processes based on combining process templates with business rules. We showed how the process materialization approach allows separation of basic process flow from business policy elements in the design of a process and also integrates resource and data needs of a process tightly. In future, we will incorporate “events”, which are also an important business process design element, into our process template design approach. In Figure 1, we showed *start* and *end* events. Events add expressive power to a process model. We plan to add events of different types and provide formal definitions for them in future work.

We are building a prototype to test and evaluate this methodology following the proposed architecture. A further challenge lies in making an interface that allows users to describe the rules in an easy way and making the details of the language completely transparent to them. We intend to explore various solutions for this including the use of an English-like rule language such as SBVR [10] and providing a GUI interface to increase ease of use and prevent typing errors. More effort will also be devoted to the semantics for rule conflict resolution. Lastly, adding ontologies to the architecture would increase the expressive power of the framework.

References

1. van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1), 21–66 (1998)
2. van der Aalst, W.M.P., et al.: Workflow Patterns. *Distributed and Parallel Databases*, 14(3): 5–51 (2003)
3. van der Aalst, W.M.P., Rosemann, M., Dumas, M.: Deadline-based escalation in process-aware information systems. *Decis. Support Syst.* 43(2), 492–511 (2007)
4. Bhattacharya, K. et al., Towards Formal Analysis of Artifact-Centric Business Process Models. In: *Business Process Management (BPM)*, Brisbane, Australia, pp. 288–304 (2007)
5. Chiu, D.K.W., Li, Q., Karlapalem, K.: Web interface-driven cooperative exception handling in ADOME workflow management system. *Web Information Systems Engineering*. 26(2), 93–120 (2001)
6. Clocksin, W.F., Mellish, C.S.: *Programming in Prolog*. Springer, New York (1987)
7. Curbera, F., et al.: Exception Handling in the BPEL4WS Language. In: *Business Process Management (BPM)*, pp. 276–290 (2003)
8. Dumas, M., van der Aalst, W.M.P., Hofstede A.H.M.: *Process Aware Information Systems*. Wiley-Interscience (2005)

9. Goedertier, S., Vanthienen, J.: Designing Compliant Business Processes with Obligations and Permission. In: Proceedings of Workshop on Business Process Design, pp. 5–14 (2006)
10. Goedertier, S., Mues, C., Vanthienen, J.: Specifying Process-Aware Access Control Rules in SBVR. In: Paschke, A., Biletskiy, Y. (eds.) RuleML 2007. LNCS, vol. 4824, pp. 39–52. Springer, Heidelberg (2007)
11. Halliday, J. J., et al.: Flexible Workflow Management in the OPENflow System. In Proceedings of the Fifth IEEE International Enterprise Distributed Object Computing Conference (EDOC'01), pp. 82–92 (2001)
12. Hwang, S.-Y., Tang, J.: Consulting past exceptions to facilitate workflow exception handling. Decision Support Systems. 37(1), 49-69 (2004)
13. JBoss Community, Drools Flow, <http://www.jboss.org/drools/drools-flow.html>.
14. Joeris, G.: Defining Flexible Workflow Execution Behaviors. In Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, GI Workshop Proceedings – Informatik, pp. 49–55 (1999)
15. Kiepuszewski, B., et al.: On Structured Workflow Modeling. In: Proceedings of the 12th International Conference on Advanced Information System Engineering (CAiSE), LNCS, vol. 1789, pp. 431–445. Springer, London, UK (2000)
16. Kumar, A., Liu, R.: A Rule-Based Framework to Design Compliant Business Processes Using Role Patterns, The International RuleML Symposium, *RULEML-2008*, LNCS, vol. 5321, pp. 58-72, Springer, Berlin Heidelberg, 2008.
17. Kumar, A., Wang, J.: A framework for designing resource driven workflow systems. The International Handbook on Business Process Management. In Rosemann M., vom Brocke J. (eds.), Springer (2009) (forthcoming)
18. Lienhard, H., Künzi, U.-M.: Workflow and business rules: a common approach. BPTrends. <http://www.bptrends.com/>
19. Mangan P., Sadiq, S.: On Building Workflow Models for Flexible Processes. In Proceedings of the 13th Australasian Conference on Database Technologies (ADC), vol. 5, pp.103-109, Melbourne, Victoria, Australia (2002)
20. Müller, R., Rahm, E.: Rule-Based Dynamic Modification of Workflows in a Medical Domain. In Buchmann, A.P. (Ed.) BTW'99, Freiburg im Breisgau. Springer, Berlin, pp. 429-448 (1999)
21. OMG, Business Process Modeling Notation (BPMN) Version 1.0. OMG Final Adopted Specification, Object Management Group (2006)
22. Reichert, M., Dadam, P.: Adept_flex—Supporting Dynamic Changes of Workflows Without Losing Control. J. Intell. Inf. Syst. 10(2), 93-129 (1998)
23. Reijers, H., et al.: Improved Model Management with Aggregated Business Process Models. Data and Knowledge Engineering, 68(2), 221-243 (2009)
24. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems. Data and Knowledge Engineering, 50(1), 9-34 (2004)
25. Rosemann, M., van der Aalst, W.M.P.: A configurable reference modeling language. Information Systems. 32(1), 1-23 (2007).
26. Weske, M.: Flexible Modeling and Execution of Workflow Activities. In: Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS), pp. 713-722 (1998)
27. XPD.L. Workflow management coalition workflow standard, <http://www.wfmc.org/xpdl.html>