Comput Syst Sci & Eng (2000) 5: 267–276
© 2000 CRL Publishing Ltd

International Journal of
Computer Systems
Science & Engineering

Dealing with workflow change: identification of issues and solutions

W M P van der Aalst* and S Jablonski†

**Department of Technology Management, Eindhoven University of Technology, PO Box 513, NL-5600 MB, Eindhoven, The Netherlands*
Email: w.m.p.v.d.aalst@tm.tue.nl
†Department of Computer Science VI, Friedrich-Alexander Universität Erlangen-Nürnberg, Martenstraße 3, D-91058 Erlangen, Germany
Email: stefan.jablonski@informatik.uni-erlangen.de

Adaptability has become one of the major research topics in the area of workflow management. Today's workflow management systems have problems dealing with both momentary changes and evolutionary changes. As a result, the workflow management system is not used to support dynamically changing workflow processes or the workflow process is supported in a rigid manner, i.e. changes are not allowed or handled outside of the workflow management system. In this paper we focus on the potential problems caused by changes. To structure these problems we use a partitioning based on five perspectives. Errors resulting from change limited to one of these perspectives and multiple-perspective errors are distinguished. To clearly identify the potential problems caused by change, we distinguish between transient and incessant errors, and syntactic and semantic errors. Based on a classification of change and potential errors we point out stepping stones for solving some of the problems.

1. INTRODUCTION

Workflow management technology aims at the automated support and coordination of business processes to reduce costs and flow times, and increase quality of service and productivity [19, 22, 23]. A critical challenge for workflow management systems is their ability to respond effectively to changes. Changes may range from momentary (ad-hoc) modifications of the process for a single customer to a complete restructuring for the workflow process to improve efficiency. Today's workflow management systems are ill-suited to deal with change; they typically support a more or less idealized version of the preferred process. However, the real run-time process is often much more variable than the process specified at design-time. The only way to handle changes is to go behind the system's back. If users are forced to bypass the workflow management system quite frequently, the system is more a liability than an asset. Therefore, we take up the challenge to find techniques to add flexibility without loosing the support provided by today's systems.

To clarify terminology, we introduce the basic workflow terms using the following five perspectives: (1) *process* perspective, (2) *organization* perspective, (3) *information* perspective, (4) *operation* perspective, and (5) *integration* perspective*. In the process perspective, workflow process definitions (workflow schemas) are defined to specify which tasks need to be executed and in what order (i.e., the routing or control flow). A task is an atomic piece of work. Workflow process definitions are instantiated for specific cases. Examples of cases are: a request for a mortgage loan, an insurance claim, a tax declaration, an order, or a request for information. Since a case is an instantiation of a process definition, it corresponds to the execution of concrete work according to the specified routing. In the organization perspective, the organizational structure and the population are specified. The organizational structure describes relations between roles (resource classes based on functional aspects)

*In this paper we will use a terminology that is slightly different from the terminology used by the authors in previous publications.

and groups (resource classes based on organizational aspects), and other artifacts clarifying organizational issues (e.g. responsibility, availability). Resources, ranging from humans to devices, form the organizational population and are allocated to roles and groups. The information perspective deals with control and production data. Control data are data introduced solely for workflow management purposes, e.g. variables introduced for routing purposes. Production data are information objects (e.g. documents, forms, and tables) whose existence does not depend on workflow management. The operation perspective describes the elementary operations performed by resources and applications. Typically, these operations are used in the process perspective to create, read, or modify control and production data in the information perspective. Most operations are (partially) implemented by applications. The integration perspective is the linking pin between the other four perspectives (see Figure 1). Tasks and workflow process definitions identified in the process perspective are (1) linked to roles, groups, and resources in the organization perspective, (2) linked to data elements in the information perspective, and (3) linked to operations in the operation perspective. Operations in the operation perspective are linked to data elements in the information perspective, etc. A workflow definition is the specification of a workflow covering all aspects. An alternative term for workflow definition is workflow type. Cases are instances of a workflow type and are handled accordingly. A workflow management system aims at supporting the five perspectives shown in Figure 1. The build-time part of the workflow management system allows for the specification of these perspectives. The run-time part of the workflow management system takes care of the actual enactment.

The workflow management systems available today assume that the five perspectives are known *a priori* (i.e. at design time) and are not subject to frequent changes. As a result, two major problems arise:

- At design time parts of the workflow are not specified (because they are not known or simply too complex) causing errors and congestion, and forcing users to work around the system at run-time.
- While the system is in production, changes occur. Typically, these changes are not anticipated at design time and result in inconsistencies, breakdowns, and reduced quality of service.

These challenging problems have made adaptability one of the major research topics in the area of workflow management [3, 5–8, 11–13, 15–17, 20, 21, 24, 25, 27–29]. In this paper we will not try to solve particular aspects of the problem. Instead, we structure the possible changes using various criteria covering all perspectives. The resulting classification is used to discuss approaches to accommodate change. Thus, the paper can be regarded as a continuation of work accomplished by the authors before [2, 4, 17].

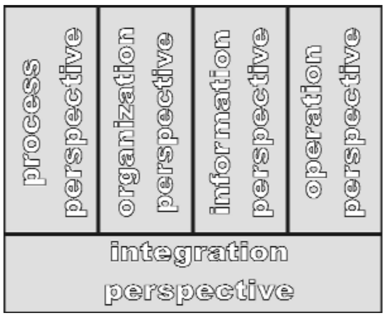

Figure 1 Five workflow perspectives

The primary focus of this paper is on correctness and consistency. We identify potential errors resulting from change. These errors are syntactic or semantic. Besides, single-perspective (e.g. a deadlock in the process perspective) and multiple-perspective (e.g. a task refers to a role that does not exist any more) errors are identified. Moreover, we distinguish between transient (temporary errors not affecting new cases) and permanent errors (also affecting new cases). In the second part of the paper, we focus on solutions for the problems identified in the first part. For this purpose, we distinguish between *flexibility by configuration* and *flexibility by adaptation*. Flexibility by configuration reduces the need for change by offering powerful design constructs. To support flexibility by adaptation we propose the use of advanced inheritance concepts. The inheritance concepts can be used to limit change, manage multiple versions/variants, and avoid errors.

2. ANATOMY OF CHANGE

To make today's workflow management systems more flexible, it is crucial to know what kinds of changes need to be supported. Note that we use the term 'change' for both (1) dealing with unanticipated events resulting from an incomplete specification, and (2) handling modifications of the specification because of changing conditions. In this section we will use six criteria to classify change.

Criterion 1 (What is the reason for change?)

1. Changes triggered by developments *outside* the system, i.e. the context/environment is the primary driver for change. Basically, there are three kind of circumstances which may trigger change:

   1.1 *Changing business context*
   The change is motivated by Business Process Reengineering (BPR) efforts, a changing marketplace (e.g., new competitors, new products), or demands of individual customers.
   1.2 *Changing legal context*
   The change is triggered by new legislature (e.g. new laws for export).
   1.3 *Changing technological context*
   New technology is available (e.g. to reduce maintenance and increase reliability) or the technical infrastructure has changed (e.g. a business partner forces the use of EDI or a service provider discontinues particular services).

2. Change triggered by developments *inside* the system. These changes are not initiated by the environment but by problems detected inside the system itself:

   2.1 *Logical design errors*
   During the design errors where made that result in run-time problems such as deadlocks or missing data.
   2.2 *Technical problems*
   The performance of the system degrades or the reliability

is threatened by failing components.

Criterion 1 locates the root of the change. The second criterion characterizes which cases are affected.

Criterion 2 (What is the effect of change?)
1. *Momentary changes* affect only one case or a selected group of cases. Changes occur on an individual or selective basis. The change is the result of an error, a rare event, or special demands of the customer. Exceptions often result in momentary changes. In general, it is not necessary to change the workflow definition, since the change will most probably not happen in this constellation again. A typical example of the root of a momentary change is the need to skip a task in case of an emergency. This change is often initiated by some external factor. A typical dilemma related to momentary change is the problem to decide what kinds of changes are allowed and the fact that it is impossible to foresee all possible momentary changes.
2. *Evolutionary changes* are of a structural nature: from a certain moment in time, the workflow changes for all new cases to arrive at the system. Existing cases (i.e., work-in-progress) may also be influenced by an evolutionary change. The change is the result of a new business strategy, reengineering efforts, or a permanent alteration of external conditions (e.g., a change of law). Evolutionary change is typically initiated by the management to improve efficiency or responsiveness, or is forced by legislature or changing market demands.

The third criterion is based on the perspectives presented in the introduction.

Criterion 3 (Which perspectives are affected?)
1. Changes in the *process perspective* affect the way cases are handled; workflow process definitions are added, deleted, or modified. Typical changes are adding tasks, deleting tasks, and changing the routing (e.g., making the process more/less parallel).
2. Changes in the *organization perspective* change the organizational structure or the population. Changes of the organizational structure are adding or deleting roles and groups and changing the relations between roles and groups. Adding or deleting resources, or changing the allocation of resources change the population.
3. The *information perspective* is changed by adding, deleting, or updating data structures (both control and production data), e.g. changing the type of a routing parameter or introducing a new form.
4. Changes in the *operation perspective* are adding, deleting, or modifying operations, e.g. a legacy application is replaced by a new application.
5. The *integration perspective* is the glue between the other four perspectives. This perspective changes if relations between the different perspectives are altered, e.g. interchanging roles assigned to tasks.

For all perspectives but the integration perspective, four types of changes are possible: extend, reduce, replace, and re-link. For the integration perspective, re-link is the only change applicable.

Criterion 4 (What kind of change?)
1. A change is of type *extend* if new entities are introduced, e.g. a task or a role is added, or a new employee is added to the organization's population.
2. A change is of type *reduce* if parts are deleted, e.g. a task is skipped or an application is removed.
3. Changes of type *replace* are a mixture of reducing and extending a specific part. For example, a task in a process definition is replaced by another task.
4. A change is of type *re-link* if there are no entities added or deleted but merely rearranged, e.g. another organizational policy is linked to a task or tasks in a process definition are reordered.

For momentary changes, the change affects specific cases (i.e. instances) rather than the specification (i.e. workflow definition). The moment of change can be limited to the moment the case is started. For evolutionary changes, change may be restricted to new cases.

Criterion 5 (When are changes allowed?)
1. *Entry time*: the moment the case enters the specification of the perspectives, it is fixed and changes are not allowed for this instance any more.
2. *On-the-fly*: changes are allowed at any point in time during workflow execution.

Both momentary and evolutionary changes are possible at entry time and on-the-fly. Customizing the process definition for a single case before the processing is started corresponds to a momentary change at entry time. If such a customization is also allowed after the processing is started, we name it an on-the-fly change. If evolutionary changes are only possible at entry time, then only the new cases that are started after the change took place have to run according to the updated workflow definition; all other cases run according to the old workflow definition. On-the-fly evolutionary changes are more difficult to handle since for each running workflow instance it must be decided how to deal with the change (cf. Criterion 6).

Changes are critical when they occur while there are cases being handled. Therefore, it is important to decide what to do with these cases (work-in-progress).

Criterion 6 (What to do with existing cases?)
1. *Forward recovery*: The old cases are aborted and appropriate measures are taken (outside the scope of the workflow management system).
2. *Backward recovery*: The old cases are aborted and rolled back or compensated, then they are restarted according to the new workflow definition.
3. *Proceed*: The old cases are handled the old way, new cases are handled the new way. There are multiple versions of the workflow and every case remains in the same version.
4. *Transfer*: The old cases are transferred to the new workflow definition.
5. *Detour*: For momentary changes it is often wise to allow a temporary detour such that the unexpected situation can be cleansed.

Note that the proceed policy corresponds to restricting change to entry time (cf. Criterion 5). The other policies correspond to on-the-fly changes.

To complete our classification of change, we discuss three terms frequently used in this context: *exception, ad-hoc workflow,* and *dynamic change*.

There is a lot of literature on exception handling [26] in the context of workflow management and extended transaction models [15, 21, 26]. Exceptions are considered to be unexpected undesirable events. Most of the results concentrate on failures of applications and networks. Deliberate change, human failures, and evolving environments are hardly taken into account. Exception handling mechanisms are primarily used to separate the failure semantics from the process logic and thus facilitate the design of readable, comprehensible workflow models. Tasks or sub-processes which fail, return an exception which is interpreted by an exception handler. The exception handler either takes action (e.g. compensate and restart or rollback) or propagates the exception to a higher level. Although exceptions fit into our framework, they are not of primary interest for this paper. Exceptions trigger changes of a specific nature. If we use the criteria to characterize the change triggered by a typical exception, then the following keywords are appropriate: inside (criterion 1), momentary (criterion 2), information/operation (criterion 3), replace (criterion 4), on-the-fly (criterion 5).

Ad hoc workflows are workflows that are defined in an *ad hoc* fashion by the end-user of the system before they are going to be executed. Workflow management systems such as Ensemble (FileNet) and InConcert (InConcert) support ad-hoc workflow. We omit-ted ad-hoc workflows because, in our opinion, it is not possible to handle ad-hoc workflows adequately by the end-user. This is a consequence of the following observations: (1) modeling workflow types is a difficult and time con-suming task, and (2) only qualified persons are able to perform the modeling task. Therefore, the assumption that an end-user of a workflow management system is able to (re-)model a workflow type in an on-the-fly manner is not justifiable. Consequently, we only consider momentary *(ad hoc*) changes. These changes should be supported in a controlled manner to avoid serious run-time errors. Among other things, a special editor is needed that guides the process of re-modeling. Section 4 discusses this issue.

The term ‘dynamic change’ refers to the problem of handling old cases in a new workflow process definition, e.g., how to transfer cases to a new, i.e. improved, version of the process. Figure 2 illustrates the dynamic change problem[†]. If the sequential workflow process (left) is changed into a workflow process where tasks *B* and *C* can be executed in parallel (right) there are no problems, i.e. it is always possible to transfer a case from the left to the right. The sequential process has five possible states and each of these states corresponds to a state in the parallel process. For example, the state with a token in *s3* is mapped onto the state with a token in *p3* and *p4*. In both cases, tasks *A* and *B* have been executed and *C* and *D* still need to be executed. Now consider the

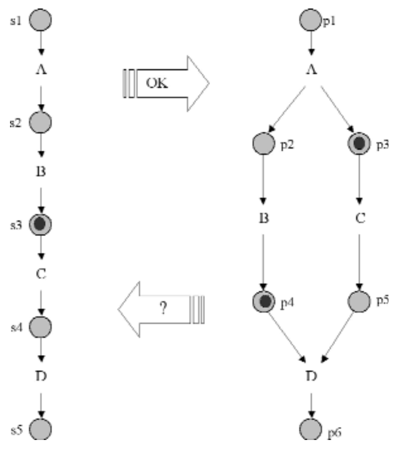

Figure 2 The dynamic change problem

situation where the parallel process is changed into the sequential one, i.e. a case is moved from the right-hand-side process to the left-hand-side process. For most of the states of the right-hand-side process this is no problem, e.g. a token in *p1* is moved to *s1*, a token in *p3* and a token *p4* are mapped onto one token in *s3*, and a token in *p4* and a token *p5* are mapped onto one token in *s4*. However, the state with a token in both *p2* and *p5* (*A* and *C* have been executed) causes problems because there is no corresponding state in the sequential process (it is not possible to execute *C* before *B*). The example in Figure 2 shows that it is not straightforward to migrate old cases to the new process after a change. Sometimes it is necessary to postpone the transfer to ensure the correct processing of a case.

According to the criterions identified above, dynamic change can be characterized as follows:

- It is not relevant whether the reason for a change is internal or external.
- A dynamic change is evolutionary; i.e. it modifies the workflow definition.
- A dynamic change can have an impact on all perspectives, especially the integration perspective.
- The kind of change is not relevant.
- Only on-the-fly changes have to be investigated.
- Existing cases have to be transferred according to the new workflow definition.

3. CORRECTNESS ISSUES

In this section we focus on the potential problems caused by change. Changing a process definition can put all cases into a deadlock. Changing the organizational model can cause starvation of the cases in the system while executing the change. Modifying data structures in the information perspective can lead to loss of work or incomplete data. Transferring a case from one process to another can lead to the unintentional double execution of a task (e.g. paying for goods) or the skipping of a vital task (e.g. sending the bill). These examples illustrate that change can cause many types of errors. One could say that without certain precautions the remedy (change) is likely to be worse than the disease (need for change).

[†]In this paper we use Petri nets to illustrate the process-related concepts. In fact, we use a restricted class of Petri nets, the so-called WF-nets [1, 2]. In a WF-net there is one source place and one sink place and all other nodes are on a path from source to sink. Readers not familiar with Petri nets and workflow modeling are referred to [2,14,19].

The costs of handling errors resulting from change can be gigantic. One error can affect many cases. Some tasks cannot be undone, e.g. transferring money or sharing information. If a task is not executed properly (i.e. during the execution there was a failure or the task should not have been executed at all), there are several countermeasures: abort, restart, rollback, compensate, rollback/restart, compensate/restart. By aborting, the remainder of the process is handled outside of the workflow management system. Restarting a task leads to loss of work. If the effect of an incorrect or inappropriate execution of a task can be undone, a rollback is issued. If the effect cannot be undone, then it may be possible to compensate (e.g. to counterbalance an inappropriate transfer of money to a supplier by subtracting the amount from the next payment). Many of these countermeasures require human intervention causing stagnation, frustration, and reduced quality of service.

To get more grip on the potential problems we classify the errors using the following criteria:

- Type of error: *syntactic* or *semantic*.
- Duration of error: *transient* or *permanent*.
- Scope of error: *single-perspective* or *multiple-perspective*.

In the remainder, we describe these criteria and give examples for each category.

3.1 Syntactic versus semantic errors

Syntactic correctness is *independent of the context*, i.e. it refers to the minimal requirements any workflow should satisfy. For example, there should be no tasks without input places. Note that syntactic correctness not only refers to the structure of the workflow process definition but also to the dynamic behavior and other perspectives. Examples of syntactic errors in the organizational perspective are roles and groups without any members and cyclic hierarchical relations. Syntactic errors in the integration perspective are pointers to entities in another perspective which do not exist, e.g. a task points to a role which has been removed. Syntactic errors are all errors that can be detected without any knowledge of the application domain, i.e. all constructs violating universal, domain independent requirements. Therefore, it is not restricted to static properties but also includes the dynamics of the workflow. Potential deadlocks and livelocks are examples of syntactic errors in the process perspective. An important correctness criterion is the so-called soundness property that guarantees proper termination [1, 2]. Proper termination means that the state is reached with a single token in the sink place and after termination there are no dangling references.

Example 1
Figure 3 shows two workflow process definitions: the left process (a) is replaced by the right process (b). The left process is sound, i.e. it is always possible to terminate properly with a token in place *o*. Examples of occurrence sequences are AEBD, ABCD, ABFEBD, and ACBFBD. However, the new process shown in Figure 3 (b) is not sound. The execution of task *E* before task *B* will result in a deadlock because

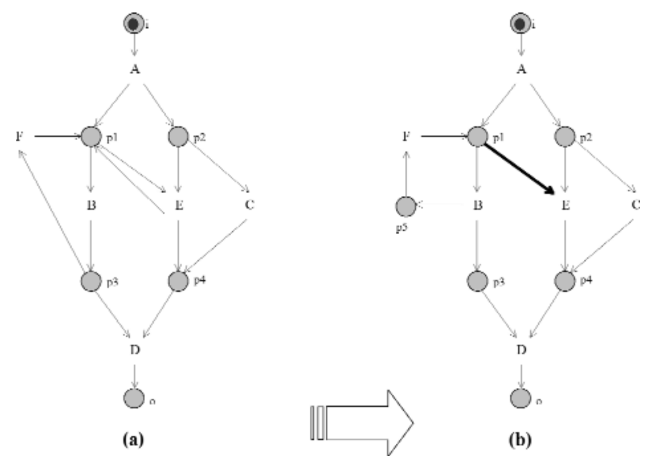

Figure 3 An example of a syntactic error

the case gets stuck in the state with just a token in *p4*. If task *C* is executed, then it is possible to execute *D* but a livelock is created because it is not possible to escape from the cycle formed by *B* and *F*. After termination (i.e. putting a token in place *o*), the case will continuously switch between a state with a token in *p5* and a state with a token in *p1*, thus causing an accumulation of tokens in *p3*. The change illustrated by Figure 3 is an example of a syntactic error.

Semantic correctness is concerned with the context in which the change occurs. Intuitively, semantic correctness deals with similarities between the external properties of the old workflow and the new workflow. To detect semantic errors knowledge of the application domain is needed.

Example 2
Figure 4 shows two fragments (old and new) of a workflow process definition. The tasks *send_bill* and *send_goods* are swapped to improve customer service and all existing cases are transferred to the new process. If a case in state *s2* (i.e., a token in place *s2*) is transferred to state *s2* in the new process, then the customer receives two bills but no goods. If the case is transferred to state *s1*, the customer still receives two bills. If the case is transferred to state *s3*, the customer still does not receive any goods. Clearly the transfer needs to be postponed. Transferring the case anyway will result in a semantic error, i.e. knowledge of the meaning of the tasks and the context is required to detect the error.

To avoid semantic errors it is often desirable that the new workflow is able to handle cases the old way (but probably has some more functionality). Consider for example the two

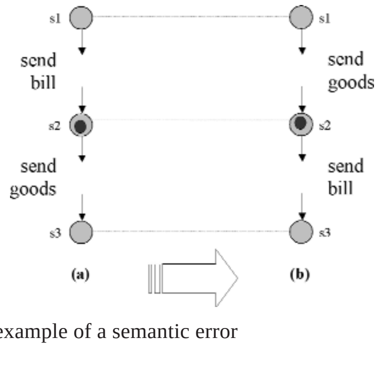

Figure 4 An example of a semantic error

process definitions shown in Figure 2. The right-hand process can do more than the left-hand process but not vice versa. Comparing different variants of the same workflow requires advanced inheritance concepts [4, 9].

3.2 Transient versus permanent errors

Some changes introduce errors which are of a temporary nature, others cause a continuous stream of errors. A transient error is a change which causes syntactic and semantic failures affecting existing cases only, i.e. new cases are not directly influenced by the error. There is a transitional period where errors occur which ends once all old cases already existing prior to the change are handled. Example 2, illustrated by Figure 4, is an example of a transient error. The following two examples show that transient errors are not restricted to the process perspective.

Example 3
To improve customer service, a company decides to introduce case management, i.e. every case has a case manager which handles (or is responsible) for handling the vital tasks for this case. The running cases do not have a case manager, therefore there is nobody to execute the tasks. A similar problem occurs if a case manager is fired. These problems are of a temporary nature because new cases are not affected.

Example 4
To speed up the access of data a new index is created using an internal identification number. A new attribute (id) is added to the set of control variables of each case. If for existing cases the new attribute is not set correctly, a transient error occurs.

Note that errors resulting from momentary changes are, by definition, transient. Evolutionary changes can cause both transient and permanent errors.

A permanent error affects, in principle, all new cases. There is not a transitional period, since only a new structural change can repair the error. Example 1 is an example of a permanent error because every new case can cause a deadlock or a livelock (see Figure 3).

Example 5
To allow for specialization an existing role in the organizational structure is split into two new roles. However, there is still a task referring to the old role obstructing the progress of cases. Similar problems occur if a role or group has no members. These errors are permanent since new changes are needed to correct them.

Example 6
For performance reasons a database application is moved from one server to another. Unfortunately, some tasks have hard-coded links to the old server where the application no longer resides thus causing run-time failures. Follow-up treatment is needed to remove this permanent error.

3.3 Single- vs. multiple-perspective errors

Some errors resulting from a change are restricted to one perspective whereas others involve consistency problems between the different perspectives. Example 1 is a *single-perspective error* since it addresses only the process perspective. Examples 2, 3 and 4 describe single-perspective errors for similar reasons. Example 5 characterizes a *multiple-perspective error* (task referring to a role which no longer exist) and a single-perspective error (role without members). Example 6 is a single-perspective error because its scope is limited to the operation perspective. If an application fails because a data structure is changed or a table is moved, then it is a multiple-perspective error.

It is helpful to know whether an error refers to a single perspective or to multiple perspectives. Specifically, the concepts ‘flexibility by adaptation’ and ‘workflow inheritance’ which will be dealt with in the next sections can leverage on this. These two concepts of dealing with changes gain from this knowledge since adequate error treatment strategies can be chosen more appropriately.

4. TWO WAYS TO DEAL WITH CHANGE

In most situations it is very cumbersome and difficult to cope with change. So it is advantageous if change could be avoided as much as possible. Although avoiding change is a primary goal, situations will be encountered that require changing either a case or a workflow definition. Specifically the issues related to Criterion 1 reveal that generally changes cannot be avoided. Think about a change that is triggered by new legislature. Many such changes cannot be anticipated but have to be reflected seriously in case processing.

The remainder of this section can be summarized as follows. Change is considered to be harmful. Thus, the best thing to do is to prevent changes as often as possible. The second best thing to do is to prepare oneself to be able to cope with changes. These two alternative strategies will be illustrated in more detail in the following.

We have elaborated an approach that avoids changes [17]. The main idea is to provide powerful, expressive modeling constructs that bear many alternative execution paths. Having such a variety of alternative executions available, the need for change is reduced since the new way of having to execute a workflow might already be covered by the powerful, expressive modeling construct. Consider the following example that clarifies this concept. A business rule says that the tasks A, B, and C have to be executed sequentially but in any order (i.e. any interleaving is allowed as long as the tasks are not executed in parallel). Due to the lack of adequate control flow constructs the modeler has chosen $A \rightarrow B \rightarrow C$ as execution model. Looking at the executions of workflows reveals that there are huge delay times since often tasks are ready for execution but have to wait until their predecessor tasks have been finished. For example, although task C is ready, it cannot be executed because B is not done yet. Thus, the modeler changes the workflow and allows the alternative execution of $B \rightarrow C$ or $C \rightarrow B$ ($A \rightarrow (B \rightarrow C \,/\, C \rightarrow B)$). Again, experience tells that this model is still not optimal. If B or C are ready for execution before A is performed, B and C would have to be delayed. Again, the workflow definition must be changed and the change may lead to one of the problems identified in the former section.

To deal with such a situation of permanent change and in order to reflect the business processes correctly, we introduce powerful expressive control flow constructs that can be defined by the application modeler. For instance, for the above situation we introduce the SEQUENCE construct [18, 19] which bears the semantics 'execute tasks sequentially but in any order': SEQUENCE (A,B,C). It is easy to comprehend that the changes discussed above are not necessary any more. We call this concept *flexibility by selection* (or more adequate *configuration*) [17]. Flexibility by configuration supports preventive treatment of changes: many changes can be avoided since business policies can be modeled more accurately. However, this feature requires that the workflow language can be extended by application specific constructs, in the above case by a new powerful control flow construct. Flexibility by configuration reduces the need for change. However, change cannot be avoided completely. Thus, a strategy must be in place to deal with changes.

In Figure 5 three situations are depicted. At a certain time a change occurs. Case A has already been terminated before the change. The change will not have an impact on case A anymore. Case C will start after the change has occurred. Usually, this case will be processed according to the new definition of the workflow. Case B is problematic: a part of the workflow process has already been executed according to the old definition of the workflow, the rest of the workflow process must still be performed. It has to be decided whether this remaining part has to be executed according to the new or the old definition of the workflow.

In a situation as depicted in Figure 5 versions and/or variants are introduced. There are, in principle, two ways to handle this situation:

- The change produces a new definition of the workflow. From a certain point in time, all cases have to be executed according to the new definition of the workflow (cf. criterion 6). We talk of introducing a new *version* of the workflow definition.
- The change produces another valid form of workflow definition. There will be cases that still will have to follow the old definition of a workflow. But, there will also be cases that have to run according to the new definition of a workflow. A new *variant* of the workflow definition is introduced.

In both alternatives, the running cases may have to be adapted to either the new version or the new variant, i.e., potentially on-the-fly changes are needed. Introducing new variants and/or versions to support change is called *flexibility by adaptation*. In contrast to flexibility by configuration, flexibility by adaptation supports the follow-up treatment of changes. Without any doubt, flexibility by configuration is to be preferred since the workflow definition need not to be changed and thus inconsistencies cannot occur. Flexibility by adaptation has to be controlled thoroughly to avoid the problems discussed before. For this purpose, we propose an approach based on advanced inheritance notions.

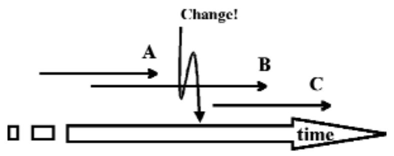

Figure 5 Changing workflows during execution

5. WORKFLOW INHERITANCE

In the previous section we concluded that flexibility by configuration is preferable over flexibility by adaptation. However, we also concluded that adaptation is unavoidable. In this paper, we have given many examples to illustrate that such adaptations can cause all kinds of problems (syntactic/semantic, transient/permanent, single/multiple-perspective). To support workflow change, we propose an approach based on inheritance. Inheritance is one of the cornerstones of object-oriented programming and object-oriented design. The basic idea of inheritance is to provide mechanisms which allow for constructing *subclasses* that inherit certain properties of a given *superclass*. In most object-oriented methods a class is characterized by a set of *attributes* and a set of *methods*. Attributes are used to describe properties of an object (i.e. an instance of the class). Methods symbolize operations on objects (e.g. create, destroy, and change attribute). The structure of a class is specified by the attributes and methods of that class.

The traditional notions of inheritance are applicable to the information and operation perspective. However, for the process perspective the traditional notions of inheritance fall short. For this perspective, a *class* corresponds to a *workflow process definition* (i.e. a workflow schema) and objects (i.e. instances of the class) correspond to *cases*. Traditional inheritance notions are restricted to the structure of a class (i.e. attributes and methods). These notions only refer to the *static* aspects of the *interface*. The *dynamic behavior* of a class is either hidden inside the methods or modeled explicitly (in UML the life-cycle of a class is modeled in terms of state machines). Although the dynamic behavior is an intrinsic part of the class description (either explicit or implicit), inheritance of dynamic behavior is not well-understood. (See [9] for an elaborate discussion on this topic and pointers to related work.) Given the widespread use of inheritance concepts/mechanisms for the static aspect, this is remarkable. Moreover, the dynamic behavior is the essence of the process perspective. In fact, dynamic behavior is the essence of workflow management. To our knowledge, the work presented in [4, 9] is the only work which deals with inheritance of dynamic behavior in a comprehensive manner. This work is based on a particular class of Petri nets: the so-called *sound workflow nets* (cf. [1,11]) mentioned earlier. This class of Petri nets corresponds to workflow processes without deadlocks, livelocks, and other anomalies. Other inheritance-based approaches abstract from the causal relations between tasks/methods.

Defining inheritance notions for workflow processes (i.e. processes defined by routing diagrams) is far from trivial. Consider two workflow processes *x* and *y*. When is *x* a subclass of *y? x* is a subclass of superclass *y* if *x* inherits certain features of *y*. Intuitively, one could say that *x* is a subclass of *y* if and only if *x* can do what *y* can do. Clearly, all tasks present in *y* should also be present in *x*. Moreover, *x* will typically add new tasks. Therefore, it is reasonable to demand that *x* can do what *y* can do with respect to the tasks present in *y*. In fact, the behavior with respect to the existing tasks should be identical. For distinguishing *x* and *y* we only consider the old tasks (i.e. the tasks already present in *y*). All other tasks are renamed to τ. One can think of these tasks as silent, internal, or not observable. Since branching bisimulation (cf. [9]) is used as an equivalence notion, we abstract from transitions with a τ label, i.e. for deciding whether *x* is a subclass of *y* only the tasks with a label different from τ are considered. The behavior with respect to these tasks is called the observable behavior. With respect to new tasks (i.e. tasks present in *x* but not in *y*) there are basically two mechanisms which can be used. The first mechanism simply blocks all new tasks and then compares the observable behavior. This mechanism leads to the following notion of inheritance.

If it is not possible to distinguish x and y when only tasks of x that are also present in y are executed, then x is a subclass of y.

Intuitively, this definition conforms to *blocking* or *encapsulating* tasks new in *x*. The resulting inheritance concept is called *protocol inheritance; x* inherits the protocol of *y*, i.e. the old routing patterns are contained in the new process. In other words, if the new tasks are not executed (i.e. blocked), one cannot distinguish any differences. Another mechanism would be to allow for the execution of new tasks but consider only the old ones.

If it is not possible to distinguish x and y when arbitrary tasks of x are executed, but when only the effects of tasks that are also present in y are considered, then x is a subclass of y.

This inheritance notion is called *projection inheritance; x* inherits the projection of the workflow process *y* onto the old tasks. Projection inheritance conforms to hiding or abstracting from tasks new in *x*. In other words, one can still enact the old routing patterns as long as one is willing to execute the appropriate new tasks.

The two mechanisms (i.e. blocking and hiding) result in two orthogonal inheritance notions. Therefore, we also consider combinations of the two mechanisms. A workflow process is a subclass of another workflow process under *protocol/projection inheritance* if by both hiding and blocking one cannot detect any differences, i.e. it is a subclass under both protocol and projection inheritance. The two mechanisms can also be used to obtain a weaker form of inheritance. A workflow process is a subclass of another workflow process under *life-cycle inheritance* if by blocking some newly added tasks and hiding others one cannot distinguish between them.

We proposed a number of inheritance preserving transformation rules [4, 9]. These rules correspond to frequently used design constructs and preserve one or more of the four inheritance notions. A detailed description of these rules is beyond the scope of this paper. Therefore, we just mention the four inheritance preserving transformation rules presented in [9]:

- Transformation rule *PT* preserves protocol inheritance and life-cycle inheritance. PT extends the superclass with new alternatives. In the resulting subclass there are alternative routes containing new tasks.
- Transformation rule *PP* preserves all four forms of inheritance, i.e. protocol/projection, projection, protocol, and life-cycle inheritance. Rule PP introduces new tasks which only postpone behavior.
- Transformation rule *PJ* preserves projection inheritance and life-cycle inheritance. Rule PJ inserts new tasks in-between existing tasks. The extension can be a single task but also a complex subflow containing many tasks and all kinds of causality relations.
- Transformation rule *PJ3* preserves projection inheritance and life-cycle inheritance. Rule PJ3 adds parallel behavior.

The rules correspond to design constructs that are often used in practice, namely choice, iteration, sequential composition, and parallel composition. If the designer sticks to these rules, inheritance is guaranteed. Moreover, if every variant/version is constructed using the four inheritance preserving transformation rules, then the transfer from one variant to another does not cause any problems, i.e., every case can be transferred without any delay and without introducing anomalies such as deadlocks, livelocks, unintended skipping of tasks, unnecessary multiple executions of common tasks, etc. See [3] for formal proofs of these statements.

The inheritance preserving transformation rules PT, PP, PJ, and PJ3 illustrate that inheritance concepts can be used to tackle the problems to workflow change. On the one hand, the rules can be used to limit change such that certain anomalies are avoided. On the other hand, the rules facilitate the management of many versions/variants (e.g. only the differences have to be stored). For the other perspectives other inheritance notions are needed. For the information and operation perspectives the traditional inheritance concepts are applicable. The organization perspective is similar to the process perspective in the sense that the traditional notions fall short. Questions such as ‘Is the new organizational structure a subclass of the old one?’ are intriguing and require further research.

Since inheritance can be used to limit change such that certain properties are preserved, it is useful for both momentary and evolutionary changes (Criterion 2), and both entry time and on-the-fly changes (Criterion 5). Each of the perspectives can benefit from suitable inheritance notions (Criterion 3). Inheritance is particularly useful for extensions of the workflow definition (Criterion 4). The inheritance preserving transformation rules mentioned in this section can be used as a carrier for transferring cases (Criterion 6) without introducing transient errors. For more information on the use of inheritance in the context of adaptive workflow, the reader is referred to [3].

6. CONCLUSION

The classification of potential errors resulting from change shows that many things can go wrong and that the problems are really persistent. Problems caused by changes in the information and the operation perspective are not new and have been a subject of extensive research, e.g. schema evolution to transfer data [10] and CORBA to hide cross machine boundaries and to facilitate upgrades. Research efforts on change in organizational perspective have been focussed mainly on organizational issues (human factors) rather than the implications on the information system. In fact, the organizational perspective has been neglected in most workflow management systems. The problems caused by changes in the process perspective have been addressed in several papers [3, 5, 7, 8, 11–13, 17, 20, 21, 24, 27, 28]. However, many problems are persistent and remain unsolved. Although multiple-perspective errors resulting from change are not very challenging from a scientific point of view (most of the problems are caused by dangling references), it is not clear how to build flexible systems to avoid such errors.

Traditional verification techniques are typically used for analyzing static designs. Evolution or momentary changes are hardly considered. Figure 6 shows that the traditional approaches are useful for finding errors which are syntactic and permanent. Given a new design, these verification techniques can be used to detect undesirable properties such as deadlocks and livelocks [1]. Transient errors are not considered because the new situation is analyzed without taking the work-in-progress into account. Semantic errors depend on the context and are difficult to detect because it is puzzling to specify the desirable properties let alone verify them. Future research should focus on verification techniques for detecting transient and/or semantic errors when changing a workflow. For a particular type of change, i.e. extending the workflow, advanced inheritance concepts seem to be useful. The inheritance-preserving transformation rules presented in [4, 9] are a good starting point for solving some the problems identified. Each of the rules corresponds to a design construct which is often used in the process perspective, namely choice, parallel composition, sequential composition, and iteration. The rules preserve to some extent syntactic and semantic correctness.

ACKNOWLEDGEMENTS

Part of this work for WMP van der Aalst was done at AFIB (University of Karlsruhe, Germany) and LSDIS (University of Georgia, USA) during a sabbatical leave.

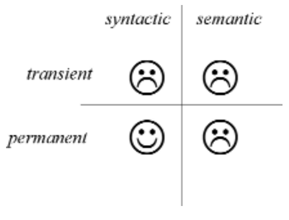

Figure 6 Traditional verification approaches can be applied to detect syntactic/permanent errors

REFERENCES

1. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azema and G. Balbo, editors, *Application and Theory of Petri Nets 1997,* volume 1248 of Lecture Notes in Computer Science, pages 407-426. Springer-Verlag, Berlin, 1997.
2. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers,* 8(1):21-66, 1998.
3. W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An approach to tackling problems related to change. *Theoretical Computer Science*, 2000 (to appear).
4. W.M.P. van der Aalst and T. Basten. Life-cycle Inheritance: A Petri-net-based approach. In P. Azema and G. Balbo, editors, *Application and Theory of Petri Nets 1997,* volume 1248 of Lecture Notes in Computer Science, pages 62-81. Springer-Verlag, Berlin, 1997.
5. W.M.P. van der Aalst, T. Basten, H.W.M. Verbeek, P.A.C. Verkoulen and M. Voorhoeve. Adaptive Workflow: On the interplay between flexibility and support. In J. Filipe, editor, *Enterprise Information Systems*, pages 63-70. Kluwer Academic Publishers, Norwell, 2000.
6. W.M.P. van der Aalst, G. De Michelis, and C.A. Ellis, editors. *Workflow Management: Net-based Concepts, Models, Techniques and Tools (WFM'98).* UNINOVA, Lisbon, June 1998.
7. W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors. *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, 2000.
8. A. Agostini and G. De Michelis. *Simple Workflow Models*. In [5].
9. T. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. PhD Thesis. Eindhoven University of Technology, Department of Computing Science, Eindhoven, the Netherlands, 1998.
10. E. Bertino, E. Ferrari, and V. Atluri. *Object-Oriented Database Systems: Concepts and Architecture*. Addison-Wesley, 1993.
11. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. *Data and Knowledge Engineering,* 24(3):211-238, 1998.
12. C. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In N. Comstock and C. Ellis, editors, *Conf. on Organizational Computing Systems*, pages 10–21. ACM SIGOIS, ACM, Aug 1995. Milpitas, CA.
13. C.A. Ellis, K. Keddara, and J. Wainer. Modeling Workflow Dynamic Changes Using Timed Hybrid Flow Nets. In [5].
14. C.A. Ellis and G.J. Nutt. Modelling and Enactment of Workflow Systems. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of Lecture Notes in Computer Science, pages 1-16. Springer-Verlag, Berlin, 1993.
15. C. Hagen and G. Alonso. Flexible Exception Handling in the OPERA Process Support System. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS),* Amsterdam, The Netherlands, 1998.
16. Y. Han and A. Sheth. On Adaptive Workflow Modeling. In *Proceedings of the 4th International Conference on Information Systems Analysis and Synthesis,* pages 108-116, Orlando, Florida, July 1998.
17. P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, and M. Teschke. A comprehensive approach to flexibility in workflow management systems. *Proceedings International Joint Conference on Work Activities Coordination and Collaboration (WACC'99)*, pages 79-88, 1999.
18. S. Jablonski. MOBILE: A Modular Workflow Model and Architecture, *Proc. of the 4th International Working Conf. on Dynamic Modeling and Information Systems*, Noordwijkerhout, NL, Sept. 1994.
19. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation* International Thomson Computer Press, 1996.
20. S. Jablonski, K. Stein, and M. Teschke. Experiences in Workflow Management for Scientific Computing. *Proceedings of the 8th International Conference and Workshop on Database and Expert Systems Applications (DEXA'97,* Toulouse, France, Sept. 1-5), 1997.
21. M. Klein, C. Dellarocas, and A. Bernstein, editors. *Proceedings of the CSCW-98 Workshop Towards Adaptive Workflow Sys-*

tems, Seattle, Nov. 1998.

22. T.M. Koulopoulos. *The Workflow Imperative.* Van Nostrand Reinhold, New York, 1995.
23. P. Lawrence, editor. *Workflow Handbook 1997, Workflow Management Coalition.* John Wiley and Sons, New York, 1997.
24. M. Reichert and P. Dadam. ADEPTflex: Supporting dynamic changes of workflow without loosing control. *Journal of Intelligent Information Systems,* 10(2):93-129, 1998.
25. A. Sheth. From Contemporary Workflow Process Automation to Dynamic Work Activity Coordination and Collaboration. *Siggroup Bulletin*, 18(3):17-20, 1997.
26. D.M. Strong and S.M. Miller. Exceptions and exception handling in computerized information processes. *ACM Transactions on Information Systems,* 13(2):206-233, 1995.
27. M. Voorhoeve and W.M.P. van der Aalst. Conservative Adaptation of Workflow. In [29].
28. M. Voorhoeve and W.M.P. van der Aalst. Ad-hoc Workflow: Problems and Solutions. In R. Wagner, editor, *Proceedings of the 8th DEXA Conference on Database and Expert Systems Applications*, pages 36-41, Toulouse, France, Sept 1997.
29. M. Wolf and U. Reimer, editors. *Proceedings of the International Conference on Practical Aspects of Knowledge Management (PAKM'96),* Workshop on Adaptive Workflow, Basel, Switzerland, Oct 1996.