Automating Handover in Dynamic Workflow Environments *

Chengfei Liu[1]** and Maria E Orlowska[2] and Hui Li[2]

[1] School of Computing Sciences, University of Technology, Sydney, NSW 2007, Australia

[2] Department of Computer Science and Electrical Engineering, The University of Queensland, Qld 4072, Australia

Abstract. Workflow technology has been widely used in business process modelling, automation and reengineering. In order to meet the fast-changing business requirements, to remain competitive in the market, an enterprise may constantly refine the workflow models of its business processes. The most challenging issue in evolution of a workflow model is the handover of its running instances from the old specification to the new specification. Such a handover depends on the semantics of a workflow model as well as the execution information of its running instances. A handover policy, therefore, needs to be specified for this purpose. In this paper, we propose a simple yet effective handover policy specification language. Using this language, a designer can easily specify a handover policy which reflect exactly what a workflow administrator needs to react to when a workflow model evolves. Criteria for the correct specification of handover policies are also addressed. Finally, a framework for automating handover of workflow instances is presented.

1 Introduction

Recent years have seen widespread use of workflow technology in business process modelling, automation and reengineering. Next only to the Internet related technology and products, the workflow technology and products [5, 11] are arguably the most influential new breed of software systems from the perspective of achieving significant impact on enterprises. Many enterprises have shifted their data-centric approach in the context of the information systems technology and solutions to a process-centric one. Workflow technology has matured to some extent, and current products are able to support a range of applications. However, many limitations remain in current workflow technology, especially for supporting more demanding applications and more dynamic environment.

* The work reported in this paper has been funded in part by the Cooperative Research Centres Program through the Department of the Prime Minister and Cabinet of the Commonwealth Government of Australia.

** Work done partially while the author was at the Distributed Systems Centre, Brisbane, Australia

In a fast-changing environment, an enterprise may constantly refine its workflow models to remain competitive in the market, to meet customers' new requirements, to change business strategies, to improve performance and quality of services, to benefit from changes in technology, etc. Two aspects are related to the evolution of a workflow model. First, the old specification of a workflow model needs to be changed to a new specification correctly. This is the static aspect of the evolution. Second, as a business process tends to be long lasting, whenever a workflow model changes its specification, there may exist a set of running instances of the old specification. How to handover these running instances is an interesting and challenging issue.

Whether or not the running instances shall evolve according to the new specification and how they evolve depend on an evolution policy which is specific to the business process which the workflow model represents for. We call this evolution policy as a *handover* policy. Currently, workflow evolution has not been sufficiently supported by workflow products. Only very primitive policies are supported by some workflow products such as *Forte Conductor* [4]. Other workflow products such as *InConcert* [6] support dynamic workflow adaptation at the instance level [14]. Schema evolution has been widely addressed in the field of Object-Oriented Databases and Software Processes [1, 16, 7]. However, little work has been done in addressing the problem of workflow evolution, particularly the dynamic aspect. In their paper [3] Casati et al. have presented a workflow modification language that supports modification of a workflow model (schema). They have also discussed the case evolution policies and have devised three main policies to manage case evolution: *abort* – to abort all running instances and to start new created instances following new specifications, *flush* – to finish all running instances first and then to allow new instances to start following new specifications, and *progressive* – to allow different instances to take different decisions. Though the progressive policy is further discussed in their paper, the fine granularity of the case evolution policy speciticatios has not been addressed.

We view a workflow model evolution as a process which consists of three steps: (1). to modify a workflow model from its old specification to its new specification. (2). to specify a handover policy for handing over the running instances of the workflow model to be evolved. (3). to apply the handover policy. A workflow model modification can be done by applying a series of modification primitives using a workflow model modification language. In our study, we focus on the handover policy which is formulated based on the old and new specifications of a workflow model. When specifying a handover policy, we assume that a specifier has knowledge of both old and new specifications of the workflow model as well as their difference. Step 1 and Step 2 are performed at build-time, while only Step 3 is performed at run-time.

The rest of the paper is organized as follows. In Section 2, we brief specification of workflow models. In Section 3, we design a handover specification language for specifying what a workflow administrator needs to react to the running instances when a workflow evolution occurs. Correct specification of handover policies is addressed in Section 4. In facilitating handover of workflow

instances, a framework for implementing the handover specification language is presented in Section 5. Section 6 concludes the paper with indication of our future work.

2 Workflow Model Specification

As the specification of a handover policy for an evolution of a workflow model is based on the old and new specifications of the workflow model, we first review the workflow modelling work. Several workflow modelling techniques have been proposed in the literature [12,2,8,13], some of them even target the transactional aspects of workflows. In [13], a graphical workflow specification language is proposed for workflow conceptual modelling. As shown in Figure 1, the language includes four types of modeling objects: task, condition, synchronizer, and flow. The flows are used to link the first three types of objects to build the specification of a workflow model.

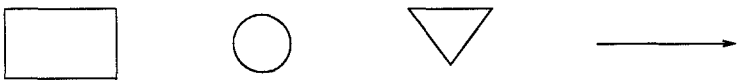

Fig. 1. Modelling Objects of Workflows

- Task – A task is a logical step or description of a piece of work that contributes towards the accomplishment of a workflow model. It can represent both automated activities and human activities. Tasks are performed by assigned processing entities. A workflow specification is basically used to specify the coordination requirements among tasks. Sometimes properties of tasks may also be specified in capturing more aspects (e.g., transactional aspects) of a workflow model. However, the actual semantics of tasks is beyond the scope of workflow specifications.
- Condition – A condition is used to represent alternative paths in a workflow specification depending on a conditional value.
- Synchronizer – At certain points in workflows, it is essential to wait for the completion of more than one execution path to proceed further. A synchronizer is used for this purpose.
- Flow – A flow defines the connection between any two objects, other than flows, in the workflow specification. It shows the flow of control or data from one object to another.

A workflow model can be represented as a *workflow graph* using these modelling objects, where nodes of the graph can be tasks, conditions and synchronizers and links of the graph are flows. Restriction is placed in constructing a correct

workflow graph. Only a limited yet relatively complete set of constructs are supported by the language. They are *Sequential, Exclusive OR-Split (Alternative), Exclusive OR-Join, AND-Split (Parallel), AND-Join (Synchronization), Nesting, Iteration, Start/Stop.* Besides, a set of correctness constraints of workflow graphs have also been identified and verification algorithms have been proposed for verifying the syntactical correctness of workflow graphs specified using this language [15, 13]. For instance, all or none of the flows proceeding a synchronizer activate for all possible instances of a workflow. Only one or none of the flows leading to a task or a condition activates for all possible instances of a workflow. The first rule eliminates the possibility of a *synchronizer deadlock.* The second rule eliminates the possibility of an *unintentional multiple execution.*

In this study, we use this graphical language to specify workflow models and assume that workflow graphs of both the old and new specifications of workflow models specified using this language are *syntactically correct.* As handover is difficult to make inside an iteration block, we treat an iteration block as a single task.

3 Handover Policy Specification

A handover policy is specified to handover current running instances of a workflow model which is to be changed. It is used to model the dynamic aspect of workflow evolution. In this section, we design a handover specification language. The objectives of the language is effective yet simple. As when a handover policy is applied to an evolution of a workflow model (i.e., from its old specification to its new specification), the running instances may be executing at any task of the old specification. What is worse, different instances can take different paths to the same task. Therefore, different instances may require different handover strategies. No matter how complex the situation can be, the language should be expressive enough for specifying all a workflow administrator wants to specify. Obviously, if all the possibilities need to be specified explicitly, it can be cumbersome, even not applicable to large workflow models. Therefore, simplification of specification must be considered. Fortunately, in practice, a workflow administrator is only interested in some key points where turning actions need to be taken. Using some default and grouping specification, the specification of a handover policy can be greatly simplified. In the following, we discuss the handover specification language.

3.1 Syntax of the Language

Associated with every workflow model evolution is one and only one handover policy. A handover policy is defined by a set of handover statements. Three handover aspects of a running instance are described in each handover statement:

- current position – indicating current executing task of a running instance;
- history – indicating the traversed paths of a running instance by conditional value;

– action – indicating the action to be taken.

A BNF definition of the handover policy specification language is given below:

```
<handover policy specification> ::= {<handover statement>}
<handover statement> ::= <on clause> <do clause>
<on clause> ::= ON <position specification>
<do clause> ::= DO <action specification> |
                IF <condition> DO <action specification> [ELSE <do clause>]
```

Position specification A position of a running instance is specified by the current executing task of that instance. In general, the exact point that the scheduler can interact is the completion point of the executing task of an instance. For the purpose of simplified specification, multiple tasks can be grouped to share a common *do clause*. Two ways of grouping are used: *tasks without order* and *tasks with order*. Position specification is further defined as follows:

```
<position specification> ::= <task name>
                           | "{"<task name> {,<task name>}"}"
                           | <task name> TO <task name>
```

Action specification In supporting handover of a running instance, two important actions must be supported. One action is *rollback*. It is used to *semantically undo* some work so that the running instance can comply with the new workflow specification. A destination task must be given for a rollback action. The other is *change-over*. It is used to migrate the execution of a running instance (or a path of it) to follow the new specification. A destination task may or may not be given for a change-over action. If a destination task is not specified in the change-over action, the task which has the same name as in the current (old) specification in the new specification is chosen as the default destination task. There is another action called *go-ahead* which may not be explicitly specified. If no handover statement is defined on a task, the default handover action after the execution of the task is going ahead. As such, the specification of a handover policy can be greatly simplified. Only the turning points need to be specified. This coincides how a workflow administrator behaves to cope with a handover.

Action specification is defined as follows:

```
<action specification> ::= ROLLBACK TO <task>
                         | CHANGE OVER [TO <task>]
                         | GO AHEAD
```

Conditional turnings Sometimes, a turning action at the current task is decided based on the history (i.e., the traversed paths) of a running instance. This is supported by the conditional turning by representing the history information in a conditional value. Besides the history information, other semantic information (e.g., time) can also be specified in the condition to facilitate flexible handover.

3.2 Handover Policy Examples

We use the PC assembling workflow example introduced in [3] to illustrate how a handover policy can be specified using our handover specification language. As shown in Figure 2, the old assembly process starts by preparing in parallel a cabinet (either a Tower or a Minitower) and a motherboard (including CPU and disk controller). Then the motherboard is inserted into the cabinet. After the FDD is inserted, a condition on *Container* is checked to determine whether a cd-rom needs to be inserted. Finally, the hard disk and video ram are inserted. The assembly process changes with the requirement that the cd-rom is replaced by NiceLab's cd-rom 4x and audio card. Based on the different decisions, different handover policies can be specified as follows.

Fig. 2. Old and New Specifications of PC Assembling Workflow Model

Example 1. Specification of *concurrent to completion* policy, i.e., all running instances are allowed to terminate following the old specification, while new instances can be started following the new specification. This policy can be expressed by the following single statement.

```
ON Start TO Plug-Video-Ram DO GO AHEAD.
```

Example 2. Specification of *Abort* policy, i.e., all running instances are aborted, and the newly created instances will start following the new specification.

```
ON Start TO Plug-Video-Ram DO ROLLBACK TO Start.
```

Example 3. Change over the running instances if executing before the condition checking on *Container*, otherwise go ahead.

```
ON Insert-FDD-1.44M DO CHANGE-OVER.
```

Example 4. Change over before the condition checking on *Container* if the instance will satisfy the condition *Container = Tower*; Rollback to the end of the task *Insert-FDD-1.44M* if the instance takes the path affected by the change.

```
ON Insert-FDD-1.44M IF Container = Tower DO CHANGE-OVER;
ON Insert-CD-Rom DO ROLLBACK TO Insert-FDD-1.44M;
ON {Add-1.6GB-HD, Plug-Video-Ram}
   IF Container = Tower DO ROLLBACK TO Insert-FDD-1.44M.
```

As shown by above examples, handover policies can be easily and directly specified using the handover specification language. A specifier only needs to explicitly specify the turning actions taken at turning points. In Example 4, a handover policy which consists of three explicit handover statements is specified. The instances executing along the path which is not affected by the change take default action, i.e., go-ahead. This example shows that arbitrary handover policies can be specified using the handover specification language.

4 Correctness Issue of Handover Policy Specification

With a handover specification language, workflow specifiers have the flexibility to support fine-granularity of handover policies. However, it may also bring the errors into specifications. As the correctness checking of a workflow model specification, it is also important to check whether a handover policy is specified correctly. As the specification of a handover policy is different from the specification of a workflow model, new correctness problems may exist for a handover policy specification.

In order to study the correctness issues of handover policy specification, we first introduce a so-called *handover graph*. Each handover policy can be defined by a handover graph. The handover graph is constructed based on the workflow graphs of both old and new specifications. Each handover statement is reflected

in the handover graph as follows:
(1). If a rollback action is defined on a task T, add a link from T to the task to which it rolls back; add a special *dead task* T_d and move all links originated from T to T_d. A dead task is a task which never gets executed.
(2). If a change-over action is defined on a task T, add a link from T to the task to which it changes over; add a dead task T_d and move all links originated from T to T_d.
(3). If a (default) go-ahead action is defined on a task T, keep the workflow graph of the old specification unchanged for T.
(4). If a conditional turning is specified, add a condition task T_c with two links originated from T_c specifying the two exclusive condition values. Depending on the turning action change the graph accordingly.

Example 5. The handover graph for the policy defined in Example 4 is given in Figure 3.

4.1 Syntactical Error Types

As the handover graph for a workflow model evolution is constructed based on workflow graphs of the old and new specifications of the workflow model and these workflow graphs are assumed syntactically correct, a handover graph can be erroneous only if the turning actions are specified incorrectly. Therefore, we aim at errors resulted from incorrect specification of turning actions. The error types which we have identified in specifying a handover policy include:

- cyclicness
  If a cycle appears in the handover graph of a handover policy, the running instances in the cycle will execute endlessly. Such a cycle must be avoided during specification of a handover policy. In Example 4, if we change the condition of either the first or the third handover statement to $Container = Minitower$, a cycle will occur and the execution will never stop.
- deadlock
  For a synchronizer node of the handover graph of a handover policy, if a dead task appears in one incoming path but does not appear in another path, a deadlock problem exists for the handover policy specification. For example, if we change over one branch of a parallel construct while keeping another branch go ahead, a deadlock will occur. Another example of deadlock can be resulted from structure mismatch. If we want to change over branches of an alternative construct to a parallel construct, a deadlock will happen to the new specification.
- unintentional multiple execution
  Similar to a workflow model specification, an unintentional multiple execution error may occur in a handover policy specification. One such example may come from changing over multiple parallel branches directly or indirectly to the same task of the new specification. Another example can be changing over different branches of a parallel construct to different branches of an alternative construct, respectively.

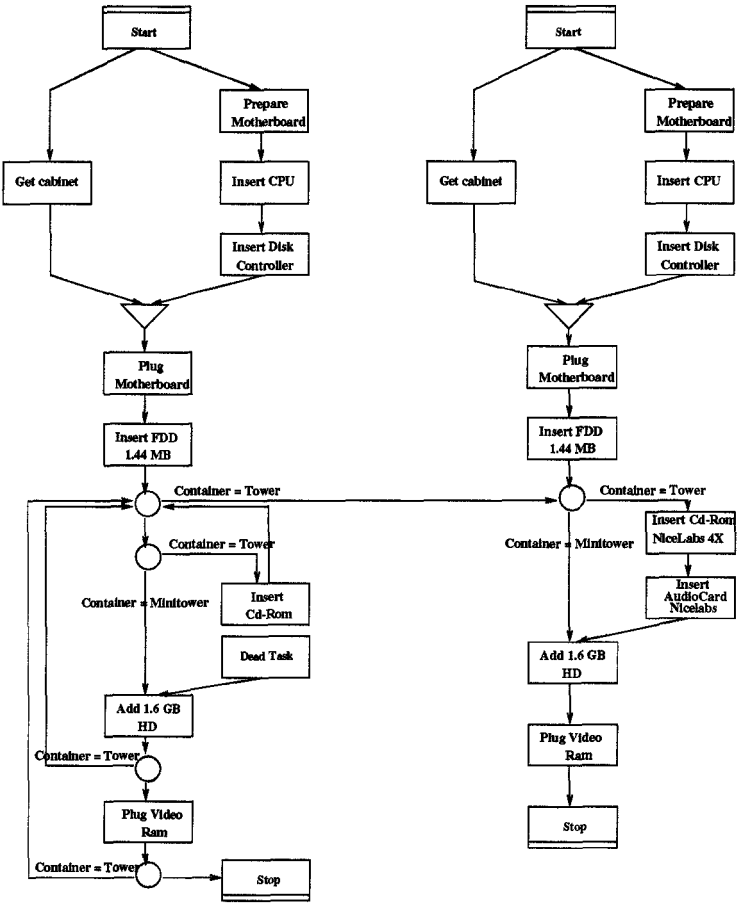

Fig. 3. A Handover Graph

4.2 Correctness Criteria for Handover Policies

In preventing handover policy specification errors, we define a set of correctness constraints as follows.

Rule 6. At most one handover action may be executed for each task of a running instance. In other words, if a cycle appears in the handover graph of a handover policy, then the conjunction of conditions specified in all condition nodes along the cycle must be *false*, i.e., the cycle is a pseudo cycle and no real cycle is allowed in the handover graph.

Rule 7. Either all branches or no branches of a parallel construct of the old specification are changed over to new specification. In other words, either all dead tasks or no dead tasks can be connected to a sychronizer.

Rule 8. Branches of an alternative construct in the old specification cannot be changed over to branches of a parallel construct in the new specification.

Rule 9. Branches of a parallel construct in the old specification cannot be changed over to branches of an alternative construct in the new specification.

Rule 10. Branches of a parallel construct in the old specification cannot be changed over directly or indirectly to the same task of the new specification.

These rules are verified for each handover policy specification before it is applied to running instances, thus run-time handover errors can be greatly reduced.

5 Facilitating Handover Policies

As automatic handover of running workflow instances has not been addressed by existing workflow management systems, it is ideal to put forward a framework which can facilitate handover based on current workflow technology. In this section, we address some key technical points towards such a framework.

5.1 Required Data Structures

The data structure for workflow instances is designed as follows:

$$WFInst(InstID, State, History, ActivePath(SpecID, CurrentPosition))$$

Where *InstID* is used for identifying a workflow instance. *State* records the current state of the workflow instance, such as, *executing, completed.* Two additional states *migrating* and *migrated* are introduced for handover purpose. The *migrating* state indicates that the workflow instance is under a handover process, when the handover process is finished, the state of the workflow instance is changed to the *migrated* state (not turning back to the *executing* state). The *migrated* state indicates that the instance needs to be treated specially in case rollback or another handover (due to newer specification or version of its workflow model) may be required to the instance later since it has undergone a handover process. *History* records the log data of all traversed paths of the workflow instance. A workflow instance may contain several active parallel paths. The number of paths will increase after an AND-Split is reached and will decrease after a synchronizer (AND-Join) is reached. Every active path has a *SpecID* and a *CurrentPosition* associated with it. A *SpecID* is used for identifying the specification on which the execution of that active path is based. During handover, it is possible that one active path is running on the old specification while another is running on the new specification. A *CurrentPosition* records the currently executing task of that path.

In addition, a new data structure for policy specification is designed.

$$Policy(SpecID, NewSpecID, Turning(Task, Condition, Action, Destination))$$

Where *SpecID* and *NewSpecID* are used for identifying the old and new workflow specifications on which a handover policy specification is based. As one and only one policy is associated with each workflow model evolution, a policy can be identified by a *SpecID*. Several turnings can be described in a policy. Each *Turning* records a *task* – after its completion the turning will take place, a *condition* – the turning may take effect only under the condition, an *action* – either rollback or change-over, and a *destination* – indicating the destination task of the turning.

5.2 Applying a Handover Policy

To apply a handover policy to a workflow specification, a workflow system command can be issued:

$$handover(SpecID)$$

This command will automatically change the state of all running instances of the specification indicated by *SpecID* to the state *Migrating*.

5.3 Scheduling a Handover Action

When a task t indicated by *CurrentPosition* in an active path p of a running workflow instance w finishes its execution, the scheduler will schedule the next step for executing. Two cases are scheduled differently:
(1) If the instance w is in a state other than *Migrating*, the scheduler will schedule w (specifically the path p) as usual, i.e., to take the next step from t according to the specification of p indicated by *SpecID*. The scheduling information is recorded in the *History*.
(2) If the state of the instance w is *Migrating*, the scheduler will first find the policy defined on the specification of p indicated by *SpecID*. After that, it tries to match t with the *Task* in a *Turning* of the policy. if it matches one and the *Condition* in the *Turning* is satisfied, then take the *Action* in the *Turning*. Otherwise keep trying. If there is no one matches, take the default next step as usual according to the specification of p indicated by *SpecID*.

Taking a Rollback Action If a rollback action is scheduled, the path p of the workflow instance w is rolled back to the point specified by *Destination*. The rollback process may be undertaken with the help of *partial compensation* [8, 10, 9] which is another technical issue in workflow management systems. The information of the rollback action and all the rollback steps need to be recorded in the *History*. After the rollback action is completed, *CurrentPosition* is changed to hold the *Destination*.

Taking a Change-Over Action If a change-over action is scheduled, the path p of the workflow instance w is changed to run on the new specification. The point where the path continues in new specification is specified by *Destination*.

The information of w needs to be modified as follows:
(a) *SpecID* for the path p is changed to hold the new specification ID, i.e., *NewSpecID*.
(b) *CurrentPosition* for the path p is changed to hold the *Destination*.
(c) A log item indicating the change-over action needs to be added to the *History*.
(d) If all active paths of the workflow instance w are changed over to running on the new specification, the state of w is changed to *Migrated* and the state change is also recorded in the *History*.

6 Conclusion and Future Work

It is an important yet challenging topic to handover running workflow instances for enterprise computing. In this paper, we specifically addressed this topic. To support flexibility and fine-granularity of handover policy specification, we designed a simple yet effective handover policy specification language. Using this language, a designer can easily and directly specify a handover policy which reflect exactly what a workflow administrator needs to react to when a workflow model evolves. Correct specification of handover policies was also discussed. A framework for automating handover of running workflow instances was presented. In the future, we will investigate algorithms for verifying the correctness of handover policy specifications. As well we will extend our current two-version support of workflow models (i.e., old specification and new specification) to multi-version support (therefore multiple handovers of long-running instances).

References

1. J. Banerjee, W. Kim, H-J. Kim, and H. Korth. Semantics and implementation of schema evolution in object-oriented databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 311–322, 1987.
2. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Conceptual modeling of workflows. In *Proceedings of OO-ER conference*, pages 341–354. Lecture Notes in Computer Science, Vol. 1021, Springer, 1995.
3. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. In *Proceedings of the 15th ER Int. Conf.*, pages 438–455. Lecture Notes in Computer Science, Vol. 1157, Springer, 1996.
4. Forte. *Forte Conductor Process Development Guide.* Forte Software, Inc., 1994.
5. Butler Group. *Workflow: Integrating the Enterprise*, June 1996.
6. InConcert. *InConcert Technical Product Overview.* InConcert Inc., January 1997.
7. M. Jaccheri and R. Conradi. Techniques for process model evolution in EPOS. *IEEE Transactions on Software Engineering*, 19(12):1145–1156, December 1993.
8. D. Kuo, M. Lawley, C. Liu, and M. Orlowska. A general model for nested transactional workflows. In *Proceedings of the International Workshop on Advanced Transaction Models and Architectures*, pages 18–35, 1996.
9. F. Leymann. Supporting business transactions via partial backward recovery in workflow management systems. In *Proceedings of BTW'95*, pages 51–70, 1995.

10. C. Liu and M. Orlowska. Confirmation: Increasing resource availability for transactional workflows. Technical report, Distributed Systems Technology Centre, September 1997.
11. Workflow Management Coalition Members. *Glossary - A Workflow Management Coalition Specification.* Workflow Management Coalition, November 1994. Softcopy available via: http://www.aiai.ed.ac.uk/project/wfmc/.
12. M. Rusinkiewicz and A. Sheth. Specification and execution of transactional workflows. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability, and Beyond.* Addison-Wesley, 1994.
13. W. Sadiq and M. E. Orlowska. On correctness issues in conceptual modeling of workflows. In *Proceedings of the 5th European Conference on Information System*, 1997.
14. A. Sheth and K. Kochut. Workflow applications to research agenda: Scalable and dynamic workflow coordination and collabration systems. In *Proceedings of the NATO ASI on Workflow Management Systems and Interoperability*, August 1997.
15. A.H.M. ter Hofstede, Maria E. Orlowska, and J. Rajapakse. Verification problems in conceptual workflow specification. In *Proceedings of the 15th ER Int. Conf.*, pages 73–88. Lecture Notes in Computer Science, Vol. 1157, Springer, 1996.
16. R. Zicari. A framework for schema updates in an object-oriented database systems. In *Proceeding of 7th International Conference on Data Engineering*, pages 2–13, 1991.