IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Hershey PA 17033-1117, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4216

Recovery in Workflow Management Systems

Sulaiman Al-Turki
K.K.M.A., Saudi Arabia, sulaiman_alturki@hotmail.com

Danilo Montesi
Department of Computer Science, University of Bologna, Italy, motesi@cs.unibo.it

ABSTRACT

Workflow Management Systems (WFMS) have been strongly tied to the database technology by most commercial and research models. This link came without a lot of consideration to the differences between the two environments. Therefore, it is important to consider each technology's requirements during the design and implementation.

I/O automata are ideal to model Workflows, where workflows are considered as black boxes (automata) with only inputs and outputs visible to the outside world. The recovery system uses I/O automata to model recovery-workflows too. In addition, forward recovery is important to use in WFMS, because undo and rollback recoveries can only be good for DBMS.

INTRODUCTION

Most shared environments are (generally) looked at as a combination of two problems, concurrency control and recovery, which have been considered as orthogonal problems. However, the developments of recovery and concurrency control mechanisms have been done independently, which has led to incompatibility [15]. Furthermore, most models have not addressed forward recovery for logical (semantical) failure. Forward recovery is important for workflow management systems, because there are number of realistic work environments that cannot use rollback recovery mechanism, such as the drilling of a hole, which cannot be undone or rolled-back.

Hence, forward recovery is not an option to be used in a WFMS, but is considered as a must. Long running activities ([6]) are common in the workflow environment and to have the process undone or redone in case of failure is impractical and very expensive. For example, when a solicitor in a house purchase dies or retire before the completion of the house purchase process, it will be very impractical to undo all the work that has been done and then re-execute all workflows again with another solicitor. Hence, a better method is to forward the work to another solicitor and finish the house purchase.

Research in workflow management systems has been active for several years, and the need for modeling exceptions in information systems has been widely recognized. However, none of the work has produced an exception handling mechanism that suits a realistic work environment. Some current WFMS provide recovery mechanisms that aid workflow database (the movement of data items) failures, but all other failures are handled manually. In general, current commercial and research models do not have a satisfactorily failure handling mechanisms for heterogeneous and distributed environments, because advanced transactional models are more data-centric [14]. Another problem is that transactional and non-transactional workflow models that were built on database concepts have the atomicity issue. To guarantee atomicity, all shared objects have to be atomic [17]. This will be very expensive to bear in a WFMS environment, because there many cases that partial work can be acceptable.

Conventional DBMS provide a variety of recovery schemes, based on logging of update operations, periodic checkpointing of the entire database, and shadow page techniques. These techniques guarantee that transactions are executed atomically, i.e., if a crash occurs during the execution of a transaction, the system will recover to a state in which either all the updates of the transaction will be reflected (if the transaction had already been committed) or none will (if the transaction had not been committed).

Section 2 gives the reader a background on what some commercial and research WFMS models have done in the recovery area. I/O automata are introduced in Section 3. Section 4 presents the proposed recovery mechanism using I/O automata.

BACKGROUND

There are number of commercial and research workflow models that have discussed the recovery issue. This section will try to give the reader an overview of what some of the well-known models have done on recovery.

FlowMark: FlowMark is an IBM workflow product that uses persistent queuing mechanism. The persistent queuing mechanism facilitates forward recovery from system failures [3]. The message will be sent persistently until the remote site declares a successful reception of the message. However, if the information regarding the execution of an activity is lost during a recovery from a system failure, it will be the responsibility of the user to intervene and resolve this failure [12]. Semantic failure is handled through partial rollback and compensation (logical undo). Compensation of failed workflows is considered (generally) retriable until successful completion.

Action Workflow: Metro and all other Action Technologies, Inc. products log the data into SQL (Structured Query Language) server then all recoveries are done in a rollback fashion. The customer or the performer can cancel the workflow activity and the system will reinstate the previous state (undo).

WAMO: The recovery mechanism in WAMO is categorized according to the type of workflow, such as document-oriented and process-oriented workflows. The system identifies different types of failures, i.e., system failure or semantic failure. In case of system failure, the workflow recovery manager (WRM) will initiate a forward crash recovery. Any uncommitted activities are removed, which is done through rollback. It is left to the local application to handle the semantic recovery from the system crash (failure). The recovery mechanism used in WAMO is clearly drawn from the database environment, since the main emphasis was on rolling back, compensation and undoing some of the effects caused by a failed activity.

WIDE: The WIDE model is a well-structured one, which was divided into two major components (Organizational and Process models) that enable the designer to set the responsibilities of agents and the definition of workflow activities. This feature is a major advantage in the WIDE model. However, the descriptions of human agents (social agents) are modeled using transactional representations. The reason behind using transactional representation is because the WIDE model is based on a commercial database application. Another disadvantage of the WIDE model is that it uses an optimistic concurrency control, which can cause costly recoveries from workflow failures.

The recovery mechanism in the WIDE model is based on partial rollback and then re-executing [5]. The rollback mechanism is applied to return the workflow to a state identical to the previous state from the perspective of business view but not from the database view. Complete rollback is impractical, hence, savepoints are used to partially rollback and the re-execute.

This paper appears in *Issues and Trends of Information Technology Management in Contemporary Organizations,* the proceedings of the Information Resources Management Association International Conference.

I/O AUTOMATA

Several references on the subject, such as [13], [4] and [10], have defined a task (process) as a state machine (automaton) that has a behavior through a state transition diagram. State diagrams provide some advantages over other approaches, such as active rules that are widely used in the database environment. Some of the advantages are:

- Simpler to understand and design.
- Can model hierarchical structure in a natural graphical style.
- Makes updating and debugging a much simpler task.

Overview of the I/O Automata

The I/O automata was first formally introduced in [10] and [11] to handle concurrent and distributed events. Each system component is modeled with an I/O automaton, which is very similar to the traditional finite-state automaton. The resulting automaton model is used to describe correct concurrent algorithms. The I/O automata system is not just meant to receive inputs and perform some computation, but it is meant to continuously receive inputs and react to its environment. One of the advantages in using the I/O automata is that it is best used for asynchronous environments [18]. Also, the I/O automata model suits concurrent and distributed environments [16].

There are three types of actions in the I/O automata model: input, output and internal. These three types of actions are categorized into two categories:

- Actions that are within the control of the automaton (internal and output actions).
- Actions that are controlled by the environment (input actions).

An automaton can model an entire system or single component. Hence, it is important that an automaton retains some information about the system being modeled. This information is substituted with a scheduler that will give the information needed to execute each automaton.

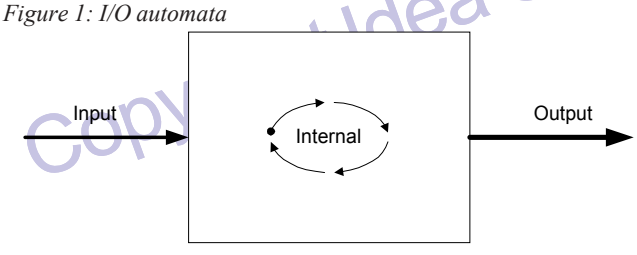
The output of automata is going to be the input of others that use the same name of the action, i.e., if À is an output action of an automaton, it is going to be the input action for other automata that uses À. This means that an automaton generates autonomously and instantaneously output actions to all automata that use the same action as an input. Hence, each automaton is required to respond to each input. This property is used to define the composition of the I/O automata by having several automata yield a single automaton.

One of the advantages of the I/O automata model is that it allows description of algorithms and systems at different levels of abstraction [8] and [9]. For example, if A2 is an image of A1 under some abstraction mapping and A2 solves a problem P, then A1 solves P. Another advantage of the I/O automata model is that it allows a simple language to describe the automata through precondition and effect specifications. Although this notation is bit more restrictive than other programming languages, the I/O automata model does not constrain the user to describe the whole system using the precondition-effect notation. The I/O automata model is a general one that can be used as the basis for concurrent programming languages.

RECOVERY IN WFMS

It is unreasonable to expect workflow systems to behave like database systems in the case of exceptions or failures. A workflow system deals with a heterogeneous environment, which has human involvement. Hence, it is necessary to model a recovery system that can handle all types of exceptions and allow different applications to handle errors internally and independently from the workflow system.

Figure 1: I/O automata



When an error or inconsistent result is produced from the execution of a workflow, there should be a preset method or tool to correct the error. Most of the previous work on workflow management systems recovery has been strongly affected by the pull of the database mechanisms, namely rollback and compensation. However, workflow management systems need a more practical approach that models recovery for work and business environments. For example, signing a contract cannot be rolled-back as if it has never been signed, because in a natural business environment there must be some forward action that needs to be taken to reimburse a client or an establishment.

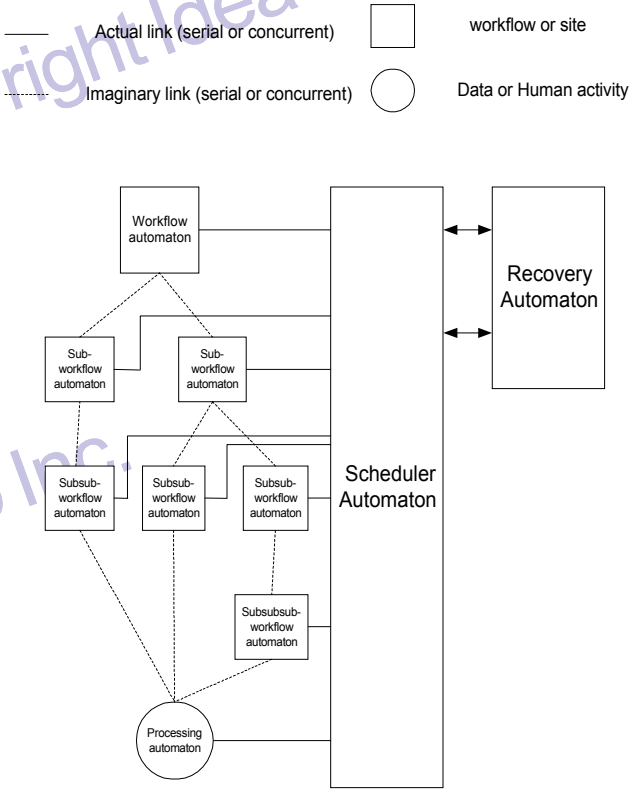
Recovery Automata

The I/O automata presented in [11] has been used in modeling workflows. Borrowing the concept of workflow automata from [1], workflows recovery is cascaded into recovery automata. In Figure 2, I/O automata represent the scheduler, all workflow trees and the recovery. The scheduler automaton interacts with the other components through input and output operations, where these operations act as the communication tool between workflow automata.

Recovery automaton is responsible for handling semantic failures that are reported by the scheduler [2]. Each categorized error or exception has a tree structure that spans in a predetermined scheme. Each component of the tree is considered as a workflow automaton. The root initiates the beginning of the recovery process.

When the scheduler receives a failure flag from one of the workflow automata, it fetches the list of semantic failures to find the appropriate response. The scheduler then will activate the appropriate recovery tree to resolve that failure. The recovery tree can in turn call on another tree to execute specific tasks.

Figure 2: Scheduler automation with recovery



Forward Recovery (Semantically)

Defining the type of recovery is as important as imposing a recovery mechanism for a WFMS. Forward recovery seems to be the ideal and most suitable mechanism for WFMS. Forward recovery is the activity of proceeding with the execution of a failed workflow without the need for rollback or re-executing while preserving the accomplishment of the failed workflow and the other workflows that depended on its results. As mentioned above there are two types of failures: System and Semantic (logical). System failures, such as a system crash from power failure, is well known and it has been investigated thoroughly in many areas. The forward recovery mechanism for a system failure means starting an execution from the last saved state. Hence, recovering from system failure is simpler than the recovery from semantic failure. WFMS needs a true forward recovery mechanism for semantic failure, because compensation and rollback in WFMS are not possible in all situations, for example, when a bullet is shot into a wall it cannot be compensated or rolled back into the gun. Although, [7] identifies this issue, no proposed solution was discussed. FlowMark model's interpretation to forward recovery is only for system failures. The compensation of semantic failure in FlowMark is done through several retries until it succeeds [3].

In many literatures the concept of recovery is done through partial rollback and re-execution. The main problem with rollback of a database transaction or a workflow is that when a transaction (T) or a workflow (Wf) is rolled back, all other transactions or workflows that used T or Wf have to be rolled back too. Hence, this is an expensive method and can lead to cascading rollbacks. Most forward recovery in the literature, such as [3] and [14], deal with system failures, which borrow the concept of persistent recovery from the operating system. Since WFMS is a heterogeneous environment, rollback or compensation is not options to be considered. In a heterogeneous environment as WFMS, compensations will try to restore database items to preserve a consistent state, but WFMS's need more than just that. In case of a logical failure, WFMS needs to preserve the state of the work environment, e.g., if a cancellation occurs, not only restoring the database as if nothing has happened but to apply any penalties that has occurred from the cancellation. Also, in WFMS there is a need to consider the fact that if a workflow fails semantically, i.e., the work cannot be continued or gives inconsistent results, a forward recovery needs be performed even if it commits. For example, if the buyer or the seller cancels a house purchase process, a forward recovery has to take care of deregistering the property and penalizing the person who cancelled the process.

Hence, WFMS needs a more realistic and practical recovery mechanism that will consider both human and application factors of the automated processes. There are number of benefits of using I/O automata to model a recovery mechanism for a WFMS:

- I/O automata give a higher level of flexibility than other recovery mechanisms, because it allows human intervention whenever needed.
- I/O automata allow specific applications to design their own recovery mechanisms independently.
- It is easy to add new exceptions (failures) or to modify old ones.

Another component need to be added to the recovery model is the use of workflow semantics. Designing a recovery system should consider and exploit the semantics of the application to reduce the cost of recovery. The forward recovery mechanism needs to consider the semantics of the workflow when building a new I/O automaton recovery tree.

Working Example

A house purchase process involves number of activities. The main ones are:

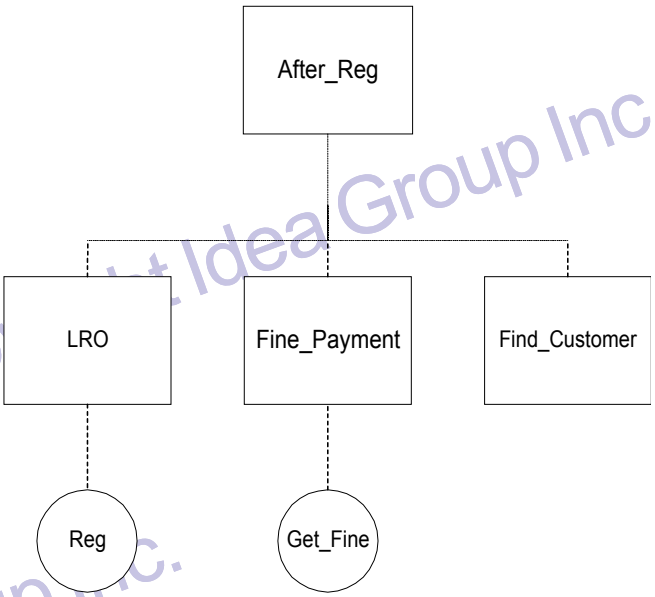
1. Collecting information on the buyer, such as name, details of the property desired, income, ...etc.
2. Collecting information on the seller, such as name, details of the property desired to sell, ...etc.
3. Getting a mortgage, which involves collecting credit background on the buyer and get the real value of the property through a surveyor.
4. The lawyer has to be contacted and it is the duty of the lawyer to prepare the contract.
5. Once all parties have signed the contract, the Land Registry Office has to be notified by the lawyer to change the property to the new owner.

If everything works in a correct order all the time, there will be no need for recovery. However, things can go wrong in reality. If the buyer withdrawals from the purchase process after the name on the property has been changed to his/her, there will be a need to activate the recovery procedure (tree). When such a withdrawal occurs, the scheduler will be notified. The scheduler then checks with the recovery automaton as to what tree should be activated.

The scheduler in this example activates the tree called After_Reg. Each box in Figure 3 resembles an automaton that is responsible for a specific activity. After_Reg automaton may acts as a Legal Department that needs to coordinate all withdrawal procedures. Once the tree has been activated, a predefined procedure is immediately followed. The After_Reg automaton will concurrently activate three siblings: LRO (Land Registry Office), Fine_Payment and Find_Customer. The LRO automaton is responsible of contacting the LRO to restore the original owner's name. The Fin_Payment automaton is responsible for finding a method of collecting the payment, such as a collection agency, from the buyer as penalty of the withdrawal. The Find_Customer automaton will communicate with the Sales Department to put the property on the market.

Each of the above mentioned automata have specified external operations (inputs and outputs) that are controlled by the scheduler automaton. However, the internal activity and the method of completing the designed task is the responsibility of the performer. For example, the LRO can use whatever method of restoring the original owner's name, such as database or manual filing system.

Figure 3: Recovery tree for canceling the purchase process



CONCLUSION

Numbers of WFMS models (commercial and research) have not paid a great deal of attention to the recovery process, because of number of reasons. One of those reasons is that recovery from failure has not been thought of until the end of the design of the model. Also, there are well-known recovery mechanisms that are widely used and can be borrowed and applied on WFMS without paying much attention to the complexity of using such mechanisms. However, neglecting the recovery mechanism can be costly to the user of the WFMS.

This paper has presented a method that can be used in designing both the WFMS and its recovery mechanism. Unlike most WFMS models, the recovery mechanism presented here has been thought of and considered during the design of the WFMS. The I/O automata give great deal of flexibility to the system and user in a structured manner. An automaton is capable of modeling a single component, such as a printer or a single human activity, or a complete system, such as a DBMS. Furthermore, the I/O automata allow system designers to use any language desired in implementing the complete WFMS or a single component.

REFERENCES

[1] Sulaiman Al-Turki and Danilo Montesi. 'Workflow in Black Boxes'. In Proceed ings of the Collaboration and Information Society Conference IS'2001, Ljubljana, Slovenia, October 22-26, 2001.

[2] Sulaiman Al-Turki and Danilo Montesi. 'Coordination in Workflow Management Systems'. In Proceedings of the 11th Workshop on Information Technologies and Systems WITS'01, New Orleans, U.S.A., 15-16 December 2001.

[3] G. Alonso, M. Kamath, D. Agrawal, A. El Abbadi, R. Günthör, and C. Mohan. 'Failure Handling in Large Scale Workflow Management Systems'. Technical report, IBM Research Report RJ9913, November 1794.

[4] Paul Attie, Munindar Singh, Amit Sheth, and Marek Rusinkiewicz. 'Specifying and Enforcing Intertask Dependencies'. In Proceedings of the Ninteenth VLDB, Irland, 1993.

[5] L. Baresi, F. Casati, S. Castano, S. Ceri, M.G. Fugini, I. Mirbel, B. Pernici, G. Pozzi, and P. Grefen. 'WIDE Workflow Development Methodology'. Technical Report 3027-6, ESPRIT Project 20280, 03-03-1999.

[6] Umeshwar Dayal, Meichun Hsu, and Rivka Ladin. 'A Transactional Model for Long-Running Activities'. In Proceedings of the 17th International Conference on Very Large Data Bases, Barcelona, Catalonia ,Spain, September 3-6, 1991.

[7] Johann Eder and Walter Leibhart. Workflow Transactions, pages 195 – 202. John Wiley and Sons Ltd., U.S.A., 1997.

[8] Alan Fekete, Nancy Lynch, and William Weihl. 'A Serialization Graph Construction For Nested Transactions'. Technical report, Massachusetts Institute of Technology, Laboratory of Computer Science, 1990.

[9] Dean Kuo. 'Model and Verification of Data Manager Based on ARIES'. ACM Transaction on Database Systems, 21(4), Dec. 1996.

[10] N. Lynch and M. Tuttle. 'Hierarchical Correctness Proofs for Distributed Algorithms'. In Proceedings of the 6th ACM Symposium on Principles of Distributed Computation, 1987.

[11] Nancy Lynch and Mark Tuttle. 'An introduction to Input/Output automata'. CWI-Quarterly, 2(3), 1989.

[12] C. Mohan, D. Agrawal, G. Alonso, A. El Abbadi, R. Günthör, and M. Kamath. 'Exotica: A Project on Advanced Transaction Management and Workflow Systems'. ACM SIGOIS Bulletin, 16(1), August 1995.

[13] Amit Sheth and Marek Rusinkiewicz. 'On Transactional Workflows'. Data Engineering Bulletin, 16(2):37–40, 1993.

[14] Amith Sheth and Devashish Worah. 'Transactions in Transactional Workflows', pages 3–34. Kluwer Publisher, U.S.A., 1997.

[15] Radek Vingralek, H. Hasse-Ye, Yuri Breitbart, and Hans-Jrg Schek. 'Unifying Concurrency Control and Recovery of Transactions with Semantically Rich Operations'. Theoretical Computer Science, 190(2), 20 January 1998.

[16] William E. Weihl. 'The Impact of Recovery on Concurrency Control'. Journal of Computer and System Sciences (JCSS), 47(1), 1993.

[17] William E. Weihl. 'Local Atomicity Properties: Modular Concurrency Control for Abstract Data Types.'. ACM Transactions on Programming Languages and Systems (TOPLAS), 11(2), April 1989.

[18] Sue-Hwey Wu, Scott A. Smolka, and Eugeune W. Stark. 'Composition and Be-haviors of Probablistic I/O Automata'. Theoretical Computer Science, 176(1–2), 1997.