Recovery Issues in Web-Based Workflow

John Miller, Amit Sheth, Krys Kochut and ZongWei Luo
LSDIS Lab, Computer Science Department
The University of Georgia
Athens, GA 30602-7404

Abstract

This paper surveys the work done by the LSDIS Lab on recovery for workflow. It then focuses on recovery issues for Web-based workflow with the WebWork enactment service used as a case study. The WebWork enactment service emphasizes ease of development of workflow applications, installation, and use. WebWork provides low overhead, yet what we believe to be effective recovery mechanisms. The paper classifies ten different type of errors or failures that one is likely to encounter in Web-based workflows and explains how WebWork deals with each of them.

1 Introduction

Workflow Management Systems (WfMSs) provide an automated framework for managing intra- and inter-enterprise business processes. According to the Workflow Management Coalition (WfMC), a Workflow Management System is a set of tools providing support for process definition, workflow enactment, and administration and monitoring of workflow processes [Hollingsworth, 1994]. WfMSs are being used today to re-engineer, streamline, automate and track organizational processes involving humans and automated information systems. This paper discusses recovery issues for the METEOR (WfMSs) [Krishnakumar and Sheth, 1995, Sheth et al., 1996]. Three runtimes or enactment services for METEOR model have been developed: ORBWork, a fully distributed CORBA-based workflow enactment system [Kochut et al., 1999], NEOWork, a CORBA-based workflow enactment system with centralized schedulers, [Xu, 1997] and WebWork, a distributed workflow enactment system relying solely on Web technology [Miller et al., 1998]. The paper briefly surveys the work done on recovery in the LSDIS Lab and then examines recovery issues for Web-based workflows in more detail with WebWork being used as a case study.

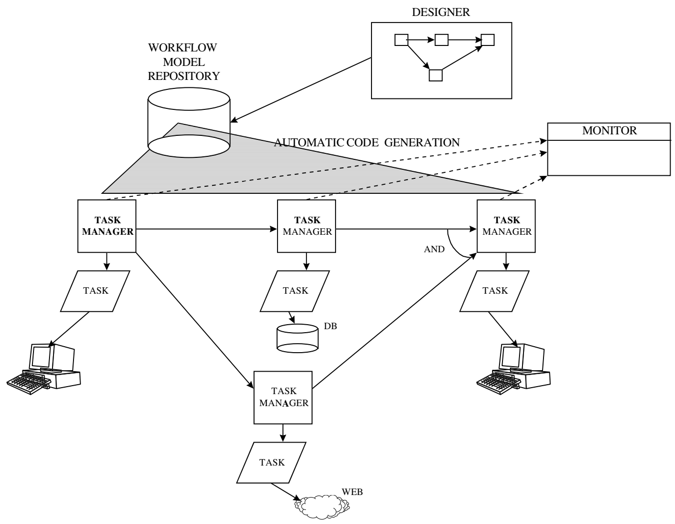

Figure 1: The METEOR Model

An enactment service provides the command, communication and control (C^3) for individual application *tasks* participating in a workflow. Tasks are the run-time instances of intra- or inter-enterprise applications. Today they typically run independently or are tied together in ad-hoc ways. WfMSs tie these tasks together in a cohesive fashion. The main components of a METEOR enactment service are workflow schedulers, task managers and (application) tasks (see Figure 1). *Task managers* as the name suggests are used to control the execution of tasks (e.g., when they execute, where they get their input, what to do when they fail and where to send their output.). To establish global control as well as facilitate recovery and monitoring, the task managers communicate with *workflow schedulers*. It is possible for schedulers to be either centralized or distributed, or even some hybrid between the two [Miller et al., 1996].

2 Error Handling and Recovery

WebWork supports the development of workflow applications that can run in heterogeneous and distributed environments. Any number of organizations can be easily incorporated into a workflow - a Web server at an organization accesses its local database(s) using CGI programs, and these CGI programs can be accessed using Web browsers at various other organizations - thus facilitating a distributed client-server implementation. In addition, multiple Web servers are needed to integrate the various autonomous organizations and databases in distributed workflow applications. CGI programs can interact with existing heterogeneous DBMSs. Thus, already existing infrastructure can be efficiently utilized. Moreover, organizations participating in a workflow can easily use existing hardware, as the Web-based system is highly platform independent.

A workflow represents a very complex computational activity. There are many tasks whose execution needs to be coordinated. Some of these tasks may be newly developed and unfortunately not tested as thoroughly as they should be. Also, resources may be temporarily unavailable and the workflows must adapt to this. Therefore, comprehensive error handling and recover mechanisms should be included with any industrial strength WfMS.

2.1 Worklists and Logs

A worklist for a manual task records the work items for that task. It keeps track of work items already processed, those being process and those already processed. (Work items processed a long time ago may be purged.)

A worklist is essential for coordinating work between two or more workflow users. A good example is an admit clerk sending patients to a nurse. This is achieved by the final task of the first user (an admit clerk) simply writing a work entry into the nurse worklist. Using a client-pull method, the first task for the next user (a nurse) periodically polls the worklist rewriting the worklist frame based on the currently available work entries.

A worklist for a task is implemented as a data structure stored with a file (although a database could also be used). Since a work item may consist of data sent from multiple tasks at different times, the data structure must merge this data into a coherent whole before the work item is ready. It must also keep track of status of work items.

For tasks that do not have a worklist, an input log is used instead. The log contains the same information and structure as a worklist, but is not used for dispensing work, since a new process can be created for each work item to be processed by an automated task.

2.2 Task Type and Recovery

The different task types are listed below in terms of the difficulty they present for recovery.

- A HUMAN_COMPUTER task is for purely manual tasks (e.g, a user fills in an HTML form). Such tasks have no application task; they are for purely human interaction. Assuming that the human participants are able to undo actions they perform outside of the workflow system, these tasks may be recovered by running the tasks again.

- A WEB task may be interactive as well as invoke an application task. These tasks provide human interaction as well as permit referentially transparent (side-effect free) application tasks to be run from a Web server. A common example is a query (read-only) to a local database. Since Web tasks are not permitted to update data, they may be re-executed.

- A TRANSACTIONAL task minimally supports the atomicity property and maximally supports all ACID (Atomicity, Consistency, Isolation, and Durability) properties [Gray and Reuter, 1993]. The application task itself is transactional, minimally supporting atomicity so that the application task either commits or aborts. A common example is a transaction on a database. If the task fails before commit, the application will automatically be undone. If the application task finishes (commits), but the task manager fails before outputting the results of the application task, the task as whole is only partially finished. To achieve atomicity during recovery, when the task manager is restarted the task wrapper checks to see if the transaction committed. If it did, the application task is skipped. This approach will not work if the update transaction also returns data. This can be dealt with by separating out the pure query from the pure update at workflow design time. Note, ORBWork will provide additional mechanisms which exploit two-phase commit as well as other global transaction protocols.

- Anything else is a NONTRANSACTIONAL task. In this case, custom coding (e.g., compensating actions) and/or expert manual intervention are required to deal with failure of these tasks.

2.3 Handling Common Error and Failures

Any WfMS must handle common errors and failures of tasks if it is to be of practical use. In addition, error handling and recovery at the task manager and workflow levels are also important [Worah and Sheth, 1997]. WebWork's goal is to provide useful and practical error handling and recovery with minimal overhead. Presently, the error handling and recovery mechanisms simply rely on three types of persistent data: (1) data maintained by the Web servers, (2) data in the WebWork worklists or input logs (one for each task), and (3) data recorded in WW_error_logs (one for each Web server).

For WebWork, we have classified ten types of errors or failures [1] that may be encountered.

2.3.1 No. 10: Data Entry Errors on Forms

An important function performed by the verifier is to check the values in data entry fields to ensure that they do not violate constraints. Typically, some of the data elements (or attributes) for a task will correspond to data entry fields in an HTML form. The verifier checks this data right at the browser where it should be checked. If the values violate the constraints, the user must re-enter the values. Constraints on data attributes or elements are gleaned from the designer specifications, or optionally, specified by the workflow developer at build time (see [Palaniswami, 1997] for details). The verifier code that performs these constraint checks is generated automatically at run-time based on these specifications. These are handled by the verifier. If the values entered violate the constraints, the page will be redisplayed with an indication of the error. Data entry fields obtain constraints in one of two ways. First, the data types specified for the attributes by the graphical workflow designer can be used to constrain the data entered. For example, if the attribute is of type int, then only digits should be entered. In addition to constraints based on data types, two other types of constraints may be specified, namely, Not-Null and Bounded. Not-Null forces the user to type something in the data entry field for the attribute, while Bounded allows lower and upper bounds to be placed on the value entered.

[1] This classification, informally known as Dev's Top Ten List, presents errors/failures in approximately increasing order of severity.

2.3.2 No. 9: Task Error

Application tasks are invoked in the execute method of its task manager. If the task fails, it will return an error code. The do_task wrapper receives this error code. The assess function determines the severity of the error. The execute method within the task manager will attempt to handle the problem and then reinvoke do_task. The mapping between error codes and actions/exceptions is purposefully set up to be very flexible. The mapping may be coarse (the default) or fine. A coarse mapping is set up by simply defining a symbol.

2.3.3 No. 8: Task Failure

Generally, running tasks as functions should be faster (no fork and exec overhead), while running them as processes should be more reliable. In both cases, signal handlers can be used to catch fatal errors (e.g., Segmentation Faults), but when an application task runs within the task manager's memory, severe damage to the task manager may have already been done before the signal is sent. This type of protection is important since task code is typically coded by the workflow application developers and as such is less likely to be rigorously tested. A task failure occurs when a task dies in the middle of its execution (e.g., because of a segmentation fault). If the task is a task function running within the address space of the task manager process, this type of failure is potentially dangerous. We deal with this by having the task manager set up signal handlers to prevent the task manager from crashing. The sig_handler writes the relevant information about the failure to WW_error_log and then may call the output fail method of the task manager or try to execute the do_task again. There is no guarantee that this will work, since the faulty task function may have wreaked havoc with the task manager's memory. A more reliable way to run an application task is as a task process. In this case, it has its own address space so that the task manager's memory will be spared any deleterious effects. The task manager detects that the task (child process) has died when its read operation on the pipe returns with an error code. At this point, the do_task function raises a TaskFailureException. Typically, the execute method will handle this exception by restoring and checking the data tuple, and re-executing do_task up to num_retries times. Since the previous call may have corrupted the data tuple, it is restored from a copy.

2.3.4 No. 7: Repeated Task Errors or Failures

If a task returns an error or fails, its task manager will try to re-execute it as discussed above. On second and subsequent retries, the task manager may sleep for increasingly longer periods of time in the hope that possible resource problems may clear up over time. After a task manager has retried a sufficient number of times (num_retries), it gives up and and records the errors in the WW_error_log. Then it transitions to the fail state which enables the next appropriate task. It is also up to the output fail method to provide an appropriate display. Successful handling of the problem is now dependent on the overall design of the workflow application. If appropriate alternative paths are designed in, then the workflow should be able to adapt to such failures.

2.4 No. 6: Task Manager Error

Even though task managers can be thoroughly tested, Murphy's Law tells us that errors will occur within task managers. The most likely errors are those involving system calls (e.g., opening files, pipes or sockets). All such suspect operations are guarded by if statements or try-catch blocks. If an error occurs, the task manager will write a message to the WW_error_log and then call the output fail method. The design of workflow may include an alternative task to carry out the workflow instance's mission, or may have a compensating task which then loops back to try the original task again.

2.4.1 No. 5: Task Manager Failure

In this case, a task manager dies in the middle of its execution. This can happen because of internal (e.g., an unhandled signal due to an internal error) or external reasons (e.g., someone with root access kills your process). If the machine running a task manager fails, clearly the task manager will fail as well, but we consider this to be a more severe failure since task managers run on machines that host Web servers. If a task manager for a manual task fails, the event will be noticed by the user since an error message will be displayed on his/her browser. If the user simply clicks reload, the Web server will run the task manager (CGI program) again. Since all inputs to the task manager have been saved in either a worklist or an input log, no information will be lost. If the task manager fails repeatedly, the user should report this to the workflow administrator. If a task manager for an automated task fails, a user may become aware of this at some later time, at which point, they should inform the workflow administrator. Some CGI program failures may be recorded in the Web server's error_log, so the workflow administrator should periodically check the log. A workflow monitor may also detect the failure of such task managers. WebWork's monitor includes an option for pushing a work item back into the active worklist so that work may be resumed at the appropriate point.

2.4.2 No. 4: Enable Failure

This happens when one task manager is unable to invoke another task manager. If this happens, reattempt to start the task manager from the display page of the previous task manager, making sure that all data entry fields contain correct data. If the error persists, the user should inform the workflow administrator. A couple of likely reasons for this is that the CGI program executable file is not available or it is not executable. If the executable file is not in the appropriate directory, the administrator can reinstall it. If it is not executable, the administrator can change its permissions. For automated tasks, such failures are likely to show up as errors on socket operations or Web server errors, so the appropriate error logs should be examined.

2.4.3 No. 3: Lost or Corrupted Input Data

It is possible that somewhere in the transmission or processing of data it gets corrupted. The TCP/IP protocol will perform such error checking for end-to-end transport. Errors will cause retransmission. At present, WebWork assumes that existing mechanisms such as those provided by TCP/IP will be sufficient. In a future version, we may add some sort of checksum mechanism of our own that computes a result based on the transmitted data. If the receiving CGI program finds a checksum error, it could use the "location" directive to redisplay the previous page. This would require output pages to be stored in files.

2.4.4 No. 2: Web Browser Failure

If a user's Web browser (e.g., Netscape Navigator) crashes, the user should simply restart the browser and run a role-initiating CGI program for one of the roles (e.g., NurseRole) s/he is authorized to perform. This program will authenticate the user and then initiate his/her interface into the workflow. This interface is displayed as a top header frame and has hyperlinks to the first task for his/her role as well as all tasks that have worklists. The user can then return to the worklist s/he was working on before the browser failed. If the browser failed will the user was working on a work item pulled off the worklist, then this work item needs to be restored. This can be done by using the monitor to undo that last entry removed from the worklist. The following types of tasks are subject to browser failures: HUMAN_COMPUTER and WEB. HUMAN_COMPUTER tasks have no application task to be concerned about, so reloading the work item presents no problem. Since application tasks for WEB tasks are restricted to actions that are referentially transparent (side-effect free), no effects of the application task need to be undone. Automated tasks (TRANSACTIONAL and NONTRANSACTIONAL) should not be hindered by a browser failure, so we need not worry about them.

2.4.5 No. 1: Web Server Failure

Web servers are rapidly becoming lifelines for organizations. Web server downtime will be problematic to WebWork as well as the organization as a whole. Consequently, Web servers should be run on reliable machines and typically dedicated to the role of providing Web service. Therefore, Web server failure should be rare. If it does happen, it is the responsibility of the Web administrator to restart the Web server. In conjunction, the WebWork administrator (may or may not be the same person as Web administrator) should restart WebWork.

3 Summary

WebWork provides low overhead, yet what we believe to be effective recovery mechanisms. We have classified ten different type of errors or failures that WebWork deals with. Handling these errors or failures is done using simple mechanisms: JavaScript verifiers, flexible error code mapping, task execution retries, signal handlers, failure transitions and dependency arcs, guarded operations on resources, logs, persistent worklists, active monitors and back/forward/reload browser operations as well as built-in capabilities of Web/Application Servers.

References

[Dogac, 1996] Dogac, A. (1996). Special-Theme Issue: Multidatabases. In *Journal of Database Management*. Idea Group Publishing, Harrisburg, PA.

[Gray and Reuter, 1993] Gray, J. and Reuter, A. (1993). *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, San Mateo, CA.

[Hollingsworth, 1994] Hollingsworth, D. (1994). The Workflow Reference Model. Technical Report TC00-1003, Issue 1.1, The Workflow Management Coalition, Brussels, Belgium.

[Kochut et al., 1999] Kochut, K. J., Sheth, A. P., and Miller, J. A. (1999). Optimizing Workflow. *Component Strategies*, 1(9):45–57.

[Krishnakumar and Sheth, 1995] Krishnakumar, N. and Sheth, A. (1995). Managing Heterogeneous Multi-system Tasks to Support Enterprise-wide Operations. *Distributed and Parallel Databases*, 3(2):155–186.

[Miller et al., 1998] Miller, J. A., Palaniswami, D., Sheth, A. P., Kochut, K. J., and Singh, H. (1998). WebWork: METEOR2's Web-Based Workflow Management System. *Journal of Intelligent Information Systems, Special Issue on Workflow Management Systems*, 10(2):185–215.

[Miller et al., 1996] Miller, J. A., Sheth, A. P., Kochut, K. J., and Wang, X. (1996). CORBA-based Run-Time Architectures for Workflow Management Systems. *[Dogac, 1996]*, 7(1):16–27.

[Palaniswami, 1997] Palaniswami, D. (1997). Development of WebWork: METEOR$_2$'s Web-Based Workflow Management System. Master's thesis, University of Georgia, Athens, GA.

[Sheth et al., 1996] Sheth, A., Kochut, K. J., Miller, J., Worah, D., Das, S., Lin, C., Palaniswami, D., Lynch, J., and Shevchenko, I. (1996). Supporting State-Wide Immunization Tracking using Multi-Paradigm Workflow Technology. In *Proc. of the 22nd. Intnl. Conference on Very Large Data Bases*, Bombay, India.

[Worah and Sheth, 1997] Worah, D. and Sheth, A. (1997). Transactions in Transactional Workflows. In *Advanced Transaction Models and Architectures*, chapter 1. Kluwer Academic Publishers.

[Xu, 1997] Xu, W. (1997). NEOWork: A Reliable CORBA-Based Workflow Enactment System for METEOR2. Master's thesis, University of Georgia. Masters Thesis.