Exception Handling in Distributed Workflow Systems Using Mobile Agents

Jiannong Cao[1] *Jin Yang*[1] *Wai Ting Chan*[1] *Chengzhong Xu*[2]

[1]Internet and Mobile Computing Lab
Department of Computing
Hong Kong Polytechnic University
Hung Hom, Kowloon Hong Kong
{csjcao, csyangj, wtchan}@comp.polyu.edu.hk

[2]Cluster and Internet Computing Lab
Dept of Electrical and Computer Engg.
Wayne State University
Detroit, Michigan 48202 USA
czxu@ns2.eng.wayne.edu

Abstract

Workflow management systems (WfMSs) are software systems used to automate, coordinate and streamline business processes of organizations. Most existing research on WfMSs has been focused on workflows based on well-structured and well-behaved business processes. Although its importance has been recognized, only recently, modeling and handling of workflow exceptions has been tackled by the workflow community. In this paper, we propose a mobile agent based approach to handling exceptions in distributed workflow management systems. Mobile agents are dispatched to find out the status of running processes in the system to keep track and troubleshoot them when necessary. We describe the classification of workflow exceptions into hierarchical levels and the corresponding design of different types of exception-handling mobile agents and their cooperation. A prototype of the mobile agent-based workflow exception handling mechanism has been implemented using the IBM Aglet platform.

1. Introduction

In recent years, workflow management has become a very popular technology for supporting business processes, in which documents, information or tasks are passed between participants according to a defined set of rules to achieve an overall business goal. A workflow management system (WfMS) is a software system mainly used to support the efficient, largely automated execution of business processes described in workflow schemas. A workflow schema represents a workflow application as a collection of tasks and their dependencies. A task is an application specific unit of activities, while dependency is a relationship between tasks that may represent notification of an event or availability of data, etc [1]. WfMSs have been used in many application domains including office automation, finance and banking, healthcare, telecommunications, manufacturing and production.

Most existing research on WfMSs has been focused on management of workflows based on well-structured and well-behaved business processes. However, handling of exceptions, that is, asynchronous and anomalous situations that fall outside the normal workflow, is a very important topic that needs to be addressed. This is especially true in a distributed workflow management system, where the order of execution of participating workflows/sub-workflows is likely in a non-determinist form. Serializing distributed workflows using locking protocol, which analog to the one used in database, on coordinating distributed workflows is unrealistic. For example when a logical problem happens in one part of the distributed workflow, other parts of the workflow are not likely notified in time so they just keep going. Even when such a problem is resolved, non-serializable execution could be resulted. Consistency, especially in the presence of concurrency, may not be guaranteed and late realization of exception is one of the key obstacles to exception handling. Furthermore, for cross-organization workflow management, each WfMS system is shielded and regarded as a local service—not accessible by outsider, and coordination between these independent inter-workflows in different systems is a problem. Tradition Ad-hoc exception handling approach is not suitable in that it depends sole on human to resolve exception, and this implies that many time-consuming human interaction between organization will be needed in resolving problems, thus, the time for realization of, and system recovery from an exception is in fact too long to be efficient. Failure in detection of exception leads to further exception propagation and further waste of resource and time.

Although its importance has been recognized, modeling and handling of workflow exceptions has been tackled only recently by the workflow community [2]. In this paper, we propose a novel approach using a group of cooperating mobile agents as an aid to handle exceptions in distributed workflow systems. Mobile agents are programs that can halt execution from a host, travel across the network, and continue execution at another host, without human interruption [3,4].

Proceedings of the 2005 IEEE International Conference on e-Business Engineering (ICEBE'05)
0-7695-2430-3/05 $20.00 © 2005 IEEE

Cooperating mobile agents are a collection of mobile agents which come together for the purpose of exchanging information or in order to engage in cooperative task-oriented behaviors [5]. Using mobile agents in managing distributed systems has several advantages. First, mobile agent technology provides an approach to overcome the difficulties that hamper tight interaction between the workflow servers. After being dispatched, the mobile agents become independent of the creating server and can operate asynchronously and autonomously. Second, because mobile agent can package a conversation and dispatches itself to a destination host, using mobile agent allows us to design algorithms that make use of the most up to date system state information for decision making. It may also lead to the reduction of the total amount of communications as the interactions can take place locally. Furthermore, mobile agent brings flexibility and scalability into distributed, dynamic systems due to its ability to encapsulate policies and algorithms and its ability to automatically tolerate transit faults and dynamic changes of the network.

In the approach of exception handling proposed in this paper, cooperating mobile agents are dispatched to find out the status of running processes in the system to keep track and troubleshoot them when necessary. A framework is developed to use mobile agents to handle exceptions in distributed workflows in real-time. This leads to the prevention of exception propagation. We describe the classification of workflow exceptions into hierarchical levels and the corresponding design of different types of exception-handling mobile agents and their cooperation. Having mobile agents working on specific part of exception handling makes it possible to have exception resolved even in the presence of concurrency. In particular, it allows dynamic and flexible synchronization of collaborating workflows. A prototype of the mobile agent-based workflow exception handling mechanism has been implemented using the IBM Aglet platform. We will also briefly describe the prototype implementation.

The rest of this paper is organized as follows. Section 2 describes background and related work in mobile agent based approaches to workflow management and exception handling. In Section 3, we describe our mobile agent-based distributed workflow system model. Section 4 describes the framework of mobile agent based exception handling. In Section 5, we present the design and implementation of a prototype of the proposed framework. Finally, Section 6 concludes the paper.

2. Background and related works

A workflow is a computerized facilitation or automation of a business process, in whole or part. It is initiated by an event, which is a request of service in the form of response from human, data, program or timer. A workflow schema defines a workflow, essentially a directed graph with nodes representing the tasks to be performed and edges representing the dependencies between the tasks. Each task is associated with one or more roles, which are the only ones authorized to execute the task. An actor (also called workflow participant) can be authorized to play several roles. Moreover, a role may be played by several actors. A workflow schema can be used to instantiate several workflows (called cases or instances).

A WfMS is a system that defines, manages, and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic. A WfMS consists of a set of workflow engines hosted inside workflow servers. A workflow engine controls process execution, which is the enactment of the workflow. It is a tool that communicates with a workflow database to store and update workflow relevant data, exercise workflow logistic and determines execution of tasks that constitute workflow.

A distributed WfMS executes workflow instances in a partitioned and distributed manner. The workflow instance is partitioned into several sub-workflows, which are executed in a distributed way on different workflow engines. The workflow engines usually run on several machines inter-connected by a computer network. The simplest approach for distributed workflow management is to control a workflow instance completely by a central workflow server. However, this approach may become impractical if there are a large number of actors and many concurrently active workflow instances, because it results in bottlenecks around the controlling server. Furthermore, the actors performing the tasks may not belong to the same organization. For example, services maybe imported from external organizations, which can be integrated into the WfMS as sub-workflows, and the WfMS of such a provider merely becomes another workflow engine in the overall distributed workflow system.

In addition to performance, there are several other reasons for developing distributed workflow management systems, including scalability, resolving heterogeneity, interoperability, better utilization of resources, and reliability [6]. In a distributed workflow management system, the workflow is partitioned and each partition is assigned to the workflow engine on a server which is located close to the potential actors of the tasks belonging to it. At run time, when the control moves from one partition to the next, the current state of the workflow instance is transferred to the workflow server of the subsequent partition. Of course, several workflow severs can execute concurrent partitions in parallel.

Mobile agent has been proposed as an approach for workflow management. Typically, they are used as case managers. Once a description is given for a process, its component activities, and a particular instance of process activation, mobile agent is deployed to control the execution of the case and acts as a workflow enactment engine [7,8]. The most widely used method is to let an agent carry with it the complete workflow schema and migrate to the necessary machines to execute the various tasks. It has the advantages such as ability to schedule dynamically, reduction of human interaction, etc. However, this single-agent approach is not suitable for implementing distributed dynamic workflows. For example, using a single agent may limit concurrency. Also, carrying the entire workflow schema is not scalable with a large and complex workflow. Therefore, in some works, multiple mobile agents are created to execute concurrent tasks in a workflow, and these agents carry only partial workflow schema with them [9].

Traditional exception handling techniques include retry, compensation alternative tasks, recovery, and termination. Used alone, they are not effective, especially for cross-organizational settings. Several works have been reported on incorporating exception handling mechanism to address this issue [10-13]. However, there is few work on using mobile agents to help design the exception handling mechanism. To the best of our knowledge, this work is the first study of making use of mobile agent for the exception handling in distributed workflow systems.

3. Mobile agent based distributed WfMS and exception handling model

In a typical distributed WfMS, several collaborating sub-workflows among organizations are viewed as one workflow. A sub-workflow is viewed as a group of tasks within each collaborating organization, and it is monitored and manipulated by the workflow engine server of each organization.

Figure 1 shows our mobile agent based distributed WfMS model, where mobile agents of different kinds and at different levels are responsible of creating and managing the execution of different parts of a workflow. The mobile agent *superSchedulerMA* is at the top level. It receives the workflow specification and then interprets it to form sub-workflow schemas. Each sub-workflow schema is assigned to a workflow engine for execution. It is managed by the next-level mobile agent, called *schedularMA* which resides on the engine. The *schedularMA* is responsible of interpreting the schema and creates corresponding tasks. The tasks are dispatched by *subWfMA* to different mobile agents called *taskResponMA*. The *subWfMA* also maintains the information about the relationship between the tasks, keeps tracking the progress of the *taskResponMAs*, and coordinates their executions. The *schedulerMA* will store a local copy of its sub-workflow schema for *subWfMA* and *taskResponMA* to query for. When a *taskResponMA* receives the assigned task, it will migrate to the target host and start the work.

All the mobile agents are capable of moving but they are with different mobility frequencies. The lower level an agent is the higher mobility it has, because lower level mobile agents (such as the *taskResponMAs*) need migrate among many hosts in a distributed environment to finish their tasks.

Except performing the workflow tasks, each mobile agent in our WfMS model is also responsible for exception handling. Each mobile agent can handle the exception in a traditional way, which is to inform the exceptions it detected to the other mobile agents of the same level or report the exceptions to the mobile agents at other levels. The mobile agent can also handle the exceptions in a special manner which is particular for our mobile agent based distributed WfMS model. A mobile agent can encapsulate the exception status and the current task execution state and migrate to other sub-workflow branches to make coordination, or migrate to other levels of the WfMS to make further processing. The predefined exception handling schemes are stored in each mobile agent, enabling the mobile agent to handle an exception efficiently. Changes to the exception handling schemes also become easy since we only need to update each mobile agent.

To sum up, there are two levels of mobility for a mobile agent in our proposed model. Except the mobility for the execution of the workflow tasks in a distributed environment, the upper level of mobility is for the exception handling when certain exception happening during the execution of the workflow task.

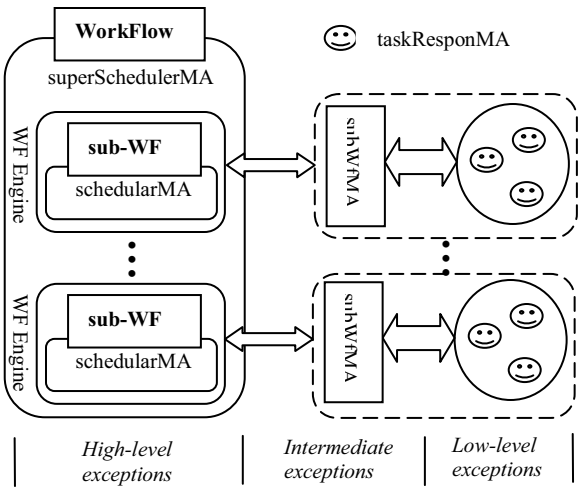

Figure 1 Mobile agent based distributed WfMS

4. Exception handling using mobile agents

Mobile agent based distributed WfMS is very suitable for exception handling due to the characteristics of mobile agent. For example, if the workflow specification has changed, e.g., a transaction needs to be eliminated due to the absence of the actor or the required resource, then all the collaborating sub-workflow schemas need to be updated to reflect the change. The first *schedularMA* who gets the notification will traverse each workflow engine to update all related sub-workflow schema so that other *schedularMAs* will dispatch and update tasks based on the most updated sub-workflow schema.

We classify the exceptions into different levels and, based on the classification, assign the corresponding mobile agents to handle different exceptions. Then, we present the design of the various exception processing functions, including exception monitoring, exception detection, and exception handling.

4.1. Classification of workflow exceptions

As shown in Figure 1, the mobile agent based distributed WfMS consists of three levels: workflow, sub-workflow, and task levels. Accordingly, exceptions can be divided into low-level exceptions, intermediate-level exceptions, and high-level exceptions.

Low-level exceptions are exceptions which could happen in executing a task, including system triggered exceptions and user triggered exceptions. Examples of low-level exceptions are: (first column is the name of exceptions, and the second column is the descriptions)

RoleUnattended	A role is not available
DataMissing	Document or data are missing
UserUnavaliable	Needed user is not available
NoResponse	No response is received
DataTypeMismatch	Data is not the expected one
TaskSeqMismatch	Task order/priority are wrong

Intermediate-level exceptions are exceptions that cannot be handled at the task level. They are closely tied to each task, or group of tasks within the sub-workflow, or even across several sub-workflows. Since *subWfMA* manages a sub-workflow, it will be responsible for these exceptions. Examples of intermediate-level exceptions are:

TaskContention	Resources contention
ConstraintViol	Constraint violation
TransacNoDef	Changes to Sub-WF schema

There are two classes of high-level exceptions: exceptions originated from the top level and errors generated from low or intermediate levels. An example of the former is the change of a work flow schema. The latter class of exceptions may be caused due to the failure of the organizations' underlying infrastructure (such as the failure of the workflow engine), or the change of the organizations' partnership. Handling of these exceptions may lead to redo, undo or reformulate the whole workflow path.

onWfRestart	WF engine restart for error
subprocessPathCross	Execution sequence wrong
ChangeWfSchema	WF schema changed by user

The three layer exceptions are not separated but are closely related. Exceptions can be propagated in both directions between lower levels and higher levels. Resolution of exceptions at a lower level can prevent it from being propagated to the higher levels.

Correspondingly, the mobile agents described in section 3 are assigned to handle the three levels of exceptions. Low-level exceptions are treated by *taskResponMA*, intermediate exceptions are handled by *subWfMA*, and high-level exceptions are handled by *superSchedulerMA* and *schedulerMA*.

The *taskResponMA* acts like an operator in each machine and handles the low-level exceptions. The *subWfMA* can be perceived as a manager in charge of group of operators, i.e. *taskResponMA*. It keeps checking the status of the whole sub-workflow by collecting the reports from *taskResponMAs*. *schedulerMA* can be viewed as an authorized administrator for the sub-workflow engine. Finally, *superSchedulerMA* can be viewed as administrator of the whole workflow, who has the ultimate right to change the execution order of sub-workflows, updating of each sub-workflow or even cancel sub-workflow. *superSchedulerMA* and *schedulerMA* together take the responsibility for high-level exceptions.

4.2. Exception Handling Functions

By providing more proactive monitoring and detection services for workflow exception handling, the proposed framework aims to assist human to resolve exceptions in a more efficient way, which can pause and restart workflow/sub-workflow at the right time before catastrophic error happens and take action to prevent exception propagation. In order to achieve this goal, we design three exception handling functions: exception monitoring, exception detection and exception handling. Exception monitoring mainly focuses on catching the exception events and then invoking the corresponding exception handlers. Exception detection attempts to check the possible exception conditions before it occur and calls the exception handling function to handle it so as to prevent the exception. In implementation, the three functions are combined to form two pairs: exception monitoring/handling and exception detection/handling.

4.2.1. Exception Monitoring/Handling. An important function in exception handling is monitoring. Monitoring functions used in conventional workflow systems is designed as a single executable component which is based on task status reported by users. Such monitoring services are undoubtedly passive, relatively simple, and human-driven. However, in modern distributed workflow systems and the transactional workflow systems, those traditional workflow-monitoring services become inappropriate in terms of the functional aspect as well as the architectural aspect. The workflow monitoring function should be active, distributed (decentralized), and automatic. Mobile agent-enabled monitoring service is suitable for distributed computing environment. First, most mobile agent platforms are Java based systems which imply the monitoring service implemented by mobile agent is platform independent. Mobile agent enabled monitoring service acts as an additional layer of monitoring service added on top of the existing one. The additional layer will function without impacting the functionalities of the existing monitoring services of the workflow management system in each organization. Second, in reality, each type of business has its own characteristics and therefore, if the business is to be carried out by a workflow management system, the system has to satisfy specific needs of the business, so the approaches to monitoring vary from business to business, so relying on the existing monitoring service of workflow engine is not sufficient. Agent-based approach allows monitoring service to be programmed such that it can be customized to serve particular business type.

At the low level, *taskResponMA* monitors execution status of workflow tasks at each machine and report them to its *subWfMA*. Low-level exceptions have to be caught immediately, because accumulation of low-level exceptions will evolve to intermediate exceptions and even harder to resolve. The *subWfMA* monitors the status of whole sub-workflow by collecting reports from *taskResponMA* and decide whether any intermediate exception has been resulted.

For low-level exceptions, the operator who monitors a task can use exception handling techniques like: pause the workflow engine (suspension sub-workflow), ignore the exception, or choose an alternative task etc, to solve a problem. However, for intermediate exceptions, due to the dependencies between each low-level exception found in each task, eliminating one or several tasks does not imply that the problem can be solved, and therefore restart of workflow engine is needed.

At the high level, *schedulerMA* and *superScheduler-MA* work together to monitor the high-level exceptions made by users. According to the source of the exception, we classify high-level exceptions into bottom-to-top exceptions and top-to-bottom exceptions according to the exception sources.

Often business partners of sub-workflows will negotiate from time to time for either modifying or terminating business logic. In this case, not only the details of each collaborating business involved will be changed, but also the partnership will be subjected to change. Top-to-bottom exceptions are changes fired from top-level decision makers down to workflow engines, which execute sub-workflow. To monitor such kind of changes, *superSchedulerMA* first checks whether the overall workflow schema has been recently changed. Once the schema has been changed, it will compare all information in the updated schema with the outdated schema, and then identify which particular piece(s) of information has been modified. After that it will traverse each sub-workflow engine and check for existing information in the schema to see whether the piece(s) of information needed to be updated found in the engine.

Bottom-to-top exceptions are something happened in task level or sub-workflow level that causes changes in the schema. We can think of *schedulerMA*, as administrator of each sub-workflow management system, who are in charge of all whole sub-workflow. It monitors the local sub-workflow schema for changes made. Once the changes happened, it will then traverse to other workflow engines to synchronize the collaborating items' detail. *schedulerMA* will also watch out if the *subWfMA* makes report about exceptions such as cancellation of activity, transaction, wrong task sequence etc.

4.2.2. Exception Detection/Handling. Exception monitoring helps to secure run-time workflow that is being executed without bypassing the exceptions. However, the preferable exception supervision should do more. For example, even the same application has been executed on the machine for many times, its performance and reliability are different at each time. It is not only the reliability problem of the application itself but also relates to the volatility of data at in a machine and the execution context. For example, the required software/hardware resources like database or human resources are unavailable. In a workflow that involves many software/hardware resources and human resources, we cannot allow the problems to be figured out at the last minute. Exception detection accomplished by mobile agents intends to solve this problem. Exception detection should be carried out throughout the workflow's execution to catch the possible exceptions before they really happen.

There are three types of "no response" exceptions that can occur at the low level. The conditions that trigger detection and the exception name, as well as the actions taken to detect the exception are listed as bellow: (LL denotes Low Level)

LL-Condition1: RoleUnattended – Request for a role's respond but get "no role assignment" response. (Possible reason is that the staff is not available now.)
Action: Before a sub-workflow begins, *taskResponMA* first checks the machine's "role assignment status" to see whether the needed role is available or not. If the expected role is not found there, an exception is detected before time-out.

LL-Condition2: DataMissing – Request for a parameter's value, but find "no such document".
Action: Existence of needed document is checked at the time when a *taskResponMA* reaches a machine for a task. If the document does not exist, an exception is detected before time-out.

LL-Condition3: NoResponse – A task waits for response and eventually time-out.
Action: The workflow management system will detect it by itself but the detection of all above exceptions will help to prevent such exception being detected by the workflow engine.

Intermediate-level exceptions are detected when certain exception pattern is received. The *subWfMA* dispatched by *schedulerMA* will send a set of *taskResponMAs* to the computers involved in the sub-workflow. Then *subWfMA* waits for the *taskReponMAs'* working status reports. The *subWfMA* analyses the reports and the arrival sequence of *taskResponMAs* to determine whether an intermediate level exception occurs. If *subWfMA* finds an exception, it will then contact either local *schedulerMA* or remote *schedulerMA* to handle the exception.

IL-Condition1: ActivityUnavaliable – Assignment of initiating role and responding role for one activity reported missing.
Action: when more than one role is reported by *taskResponMA* as RoleUnattended, *subWfMA* maps the roles into the required roles list in the definition. If the unattended roles belong to initiating role and responding role defined in the task, which implies the task is not ready to begin, this exception is detected

IL-Condition2: TaskMismatch – Received response is not the expected one.
Action: One *taskResponMA* runs at one machine and waits for response. If the task successfully finishes, the *taskResponMA* moves to *subWfMA* to report. *subWfMA* has knowledge about which *taskResponMA* should arrive. If one *taskResponMA* arrives and reports to wrong *subWfMA*, an exception is detected. It occurs when one workflow engines controls more than one sub-workflow instance and each of them interfere with each other.

IL-Condition3: TaskSequenceMismatch – Received responses have wrong sequence.
Action: *subWfMA* has the knowledge about the arrival sequence of all *taskResponMAs*. If the arriving *taskResponMAs* are out of predefined order, the *subWfMA* detects an exception. This can be due to the coordination error.

High-level exceptions are detected by both *superSchedulerMA* and *schedulerMA*. The *super-SchedulerMA* detects the changes of the schema. The changes may due to changing collaboratively-made business plan, business relationship etc., which will lead to the exceptions in whole workflow. *schedulerMA* detects changes on schema which are caused by the changes in resources, policy, and task requirement. Thos changes will produce exceptions in sub-workflows.

HL-Condition1: workflowSchemaVersionMismatch – All collaborating workflow schemas and sub-workflow schemas are supposed to be the same version. If different versions are found in the workflow schemas, it implies a mismatch.
Action: If *superSchedulerMA* detects modification made on workflow schema, an exception is detected. This can result from one schema is changed, but all the other sub-workflow schemas have not been changed.

HL-Condition2:
subworkflowSchemaVersionMismatch – Different versions in the sub-workflow schema are found, which implies there have local changes within an organization due to the change of policy
Action: *schedulerMA* detects modification made on sub-workflow schema, and then this exception detects. This can result from changing internal organization policy of one company.

Once the changes of workflow or sub-workflow schema have been detected, the exception is further derived into following three types. Activity-ModifcationException can lead to Collaboration-ModificationException, CollaborationModificationException can lead to PartnershipModificationException, and vice versa. It is because partnership can be made if collaboration is available and collaboration is defined by schema. When representatives of companies shake hands to reach a new agreement which involves creation or terminate the partnership, the running workflow engine should know at once which type of business will be affected and what tasks have to be stopped.

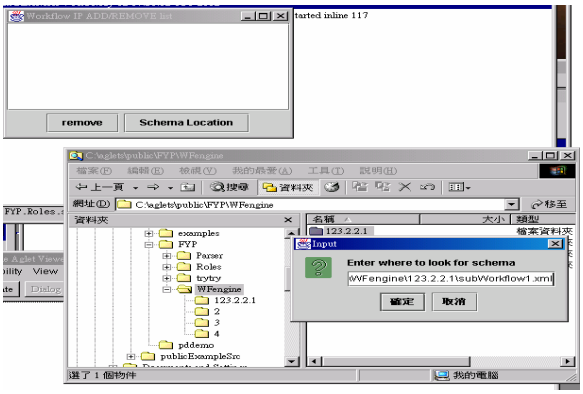

Figure 2 Moblile agent dispatching

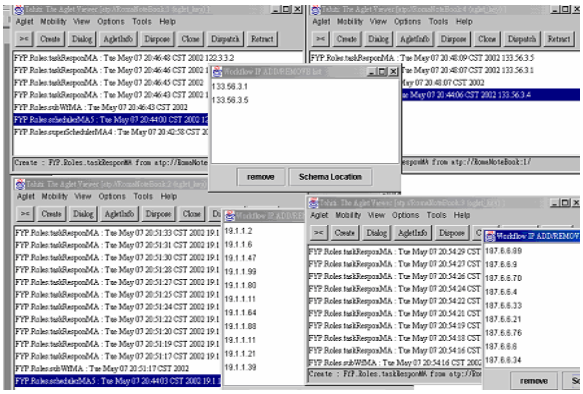

Figure 3 The execution status of four taskResoponMAs

HL-Condition3: ActivityModificationException – Service provided in a business that shared between partners is subjected to changes. Additional service adding or eliminating will result in adding activity definition or deleting activity definition.
Action: When there is a change of collaborative business plan, an exception will be detected by *superSchedulerMA* which will traverse sub-workflow engines to trace the old record. Once the old record is found, an exception is acclaimed. On the other hand, if there is a change of local organization policy or resource, *schedulerMA* will claim an exception, and traverse through sub-workflow engines one by one to update the sub-workflow.

HL-Condition4: CollaborationModificationException – Collaboration partnerships involved in one workflow are defined in workflow schema. But collaborative business can be temporarily terminated due to disagreement or expiration of partnership, which will cause the modification of the collaboration.
Action: Same as above.

HL-Condition5: PartnershipModificationException – Each partner takes at least one role in a workflow and the role can be replaced by another partner. It makes two organizations which initially do not have collaboration become partners. This will result in eliminating of partner definition in the old partner's sub-workflow schema and adding of partner definition in the new partner's sub-workflow schema.
Action: Same as above.

5. A Prototype Implementation

In order to test and evaluate our proposed system, we have implemented a prototype. Java 1.4 and IBM aglets 2.0 are used to program mobile agent, and java swing 1.1 is used to build the GUI (prompting for parameter, display working status). We use ebXML to define the workflow engine schema and Xeces is used to manipulate XML. XML-spy has been chosen as tools for editing and maintaining the ebXML-base workflow and sub-workflow schema files.

For *superSchedulerMA*, a GUI called "Workflow Schema Input" is implemented to specify the *super-SchedulerMA* the workflow schema and the set of workflow engines to be used (the set of workflow engines can also be included in the workflow schema). A workflow schema is an ebXML file which is stored on the host where the *superSchedulerMA* resides. After a workflow schema has been inputted (Figure 2), the *superSchedulerMA* will interpret the workflow schema to be a set of sub-workflow schemas for each sub-workflow. A set of *schedulerMA* are also been created and dispatched to the workflow engines together with their sub-workflow schema. The *schedulerMA* sent to remote workflow server engine will start the sub-workflow task. According to the assigned schema, *schedulerMA* creates *subWfMA* and *subWfMA* will create the related *taskResoponMA* to perform the assigned tasks. Figure 3 shows the execution status of *taskResoponMAs*. Each *taskResoponMA* can print out the exceptions it detected and the exception handling status.

In order to evaluate our system's performance, we carry out simulations on four hosts. The workflow is executed under two conditions: with or without exception handling. We assume there is no severe exceptions, that is to say, system will not restart. In our experimental workflow, we adopt an extreme exception detection mechanism that each operation will be checked before the execution. The simulation results (Table 1) show the time consumed in the execution stages for the two conditions. First, a *superScheduler-MA* will read and interpret the workflow schema and then generate a set of sub-workflow schemas and *schedulerMAs*. With exception handling functionality, a *superSchedulerMA* will keep checking the schema (the XML file). So it consumes much more time than the condition without exception handling. Since the schema checking processing is proceeding concurrently with the executions of other

mobile agents, the extra consumed time will not impact the whole workflow's execution time.

Table 1. Simulation results

Executions	Time	
	With Exception Handling	Without Exception Handling
XML(Schema) read/write	~10s	~2s
Initialization on WF engines	~5s	~5s
Average total input time	~10s	~10s
Exception detection	~8-10s	0s
Total execution time	~23-26s	~10-19s

On each host, each *taskResoponMA* will ask for some input from users and the average input time is 10 seconds. If the input time is longer than 10 seconds, the *taskResoponMA* with exception handling will assert an exception and start the exception processing (prompt an alert window to user for input and also inform upper level mobile agents to wait). Although the execution time is longer, all the mobile agents in the system can determine the exception status and make a proper judgment to proceed in the whole workflow until it is finished.

The operation of the *taskResoponMA* without exception handling depends on its timeout setting. If its timer timeouts, it will give up the execution and abort (the execution takes only 10 seconds means the workflow is abort).

6. Conclusions

In this paper, we described a mobile agent based distributed workflow management model and proposed a framework for exceptions handling using cooperating mobile agents. Exceptional situations can be very complicated. We summarize the possible exception conditions in the distributed workflow and the corresponding detection methods. Our analysis and simulations show that our model is functional and the features of being autonomous, scalable and dynamic make mobile agent a desirable choice for the exception handling of distributed workflow.

Acknowledgment

This work is supported in part by the HongKong Polytechnic University ICRG research grant G-YD63 and China National 973 Program Grant 2002CB312002.

References

[1] Workflow Management Coalition, "Terminology and Glossary", *Tech Report* WFMV-TC-1011, Workflow Management Coalition, 1996.

[2] F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi, "Specification and Implementation of Exceptions in Workflow Management Systems", *ACM Transactions on Database Systems*, Vol. 24, No. 3, Sept. 1999. pp.405-451.

[3] Danny B. Lange and Mitsuru Oshima, *"Programming and Deploying Java Mobile Agents with Aglets"*, Addison Wesley, 1998.

[4] D. B. Lange and M. Oshima, "Seven Good Reasons for Mobile Agents", *Communication of the ACM*, Vol. 42, No. 3, March 1999. pp. 88-89.

[5] J. Cao, G.H. Chan, W. Jia, and T. Dillon, "Checkpointing and Rollback of Wide-Area Distributed Applications Using Mobile Agents", *Proc. IEEE 2001 International Parallel and Distributed Processing Symposium (IPDPS2001),* April 2001, San Francisco, USA.

[6] Hans Schuster, Stefan Jablonski, Christoph BuBler, "Client/Server Qualities: A basis for Reliable Distributed Workflow Management Systems", *Proc. of the 17th Int. Conference on Distributed Computer Systems (ICDCS'97)*, Baltimore, May, 1997.

[7] Dan C. Marinescu, "Reflections on Qualitative Attributes of Mobile Agents for Computational, Data, and Service Grids", *Proc. 1st IEEE/ACM Int'l Symp. On Cluster Computing and the GRID*, 2001, pp442-449.

[8] D. M. Zimmer, "A Preliminary Investigation into Distributed Dynamic Workflow", *MS Thesis*, California Institute of Technology. 1998.

[9] V.S. Sundaram, "A Workflow Management System Using Mobile Agents", *MS Thesis*, School of Engineering, Dartmouth College, May 2000.

[10] N. Krishnakumar, and A. Sheth, "Managing Heterogeneous Multi-system Tasks to Support Enterprise-wide Operations", *Journal of Distributed and Parallel Database Systems*, 3 (2), April 1995.

[11] Z. Luo, A. Sheth, J. Miller, and K. Kochut, "Defeasible Work, its Computation and Exception Handling", *Proceedings of CSCW98, Towards Adaptive Workflow Systems Workshop*, Seattle, WA 1998.

[12] J. Eder and W. Liebhart, "Contributions to Exception Handling in Workflow Systems", *EDBT Workshop on Workflow Management Systems*, Valencia, Spain, 1998.

[13] C. Hagen, and G. Alonso, "Flexible Exception Handling in the OPERA Process Support System", *19th International Conference on Distributed Computing Systems (ICDCS)*, Amsterdam, The Netherlands, May 1998.