**Exception Handling with Workflow Evolution in ADOME-WFMS: a Taxonomy and Resolution Techniques**

Dickson K.W. CHIU, Kamalakar KARLAPALEM Qing LI

Department of Computer Science Department of Computer Science

Hong Kong University of Science and Technology City University of Hong Kong

Clear Water Bay, Kowloon, Hong Kong Tat Chee Ave, KLN, Hong Kong

Email: {kwchiu,kamal}@cs.ust.hk qli@acm.org

**Abstract:** Workflow Management Systems (WFMSs) facilitate the definition of structure and decomposition of business processes and assists in management of coordinating, scheduling, executing and monitoring of such activities. As such, the capabilities of accommodating exceptions in WFMSs are essential and of great importance. This paper first presents a taxonomy of different types of exception and their handling approaches in a WFMS based on ADOME - an active OODBMS extended with role facilities for supporting the dynamic features required for online exception handling. We then advocate a novel solution based on workflow evolution techniques. An experimental prototype system of ADOME-WFMS is currently being developed on top of the ADOME system.

1. Introduction

Many application domains like office automation, decision support systems, flexible-manufacturing systems, require complex interactions and cooperation among humans and sub-systems for facilitating day to day business processes. The work pattern is neither routine nor totally ad-hoc but somewhere in between, which calls for Next Generation Information Systems (NGIS) to be developed to streamline the automation of such application domains [LiLoc93]. A Workflow Management System (WFMS) is a system that assists in the automation of specification, decomposition, coordination, scheduling, execution, and monitoring of these activities for various business processes. Besides streamlining and improving routine business processes operations and resource management, WFMSs help in documenting and comprehending business processes. In this paper, the targeted WFMS is for handling NGIS where effective and efficient coordination of human resources and activities (as opposed to computer-based ones in existing WFMS for routine office operations) is often the key to success of modern organizations.

Since it is not possible for the designer to specify all the possible outcomes and alternatives (especially with various special cases and unanticipated possibilities), exceptions should be expected to occur frequently. A comprehensive WFMS should be able to automate exception handling by supporting the users to reallocate resources (data / object update) or amend workflows such as adding alternatives (workflow evolution). Further, frequent occurrence of similar exceptions can be incorporated into the workflow as expected exceptions. On the other hand, workflow evolution may help avoid frequent/unnecessary exceptions by eliminating error-prone activities, adding more alternatives and by enhancing the operation environment. This can lead to a WFMS that supports workflow adaptation through exceptions.

In order to support for a versatile WFMS with exception handling and workflow evolution, we attempt to address these problems by employing the following 3 major measures:

1. **Advanced modeling** - Many of the earlier WFMSs [NATO97] were built with more traditional database technologies (e.g., relational databases), and fall short in one way or another in facilitating / offering flexibility of modeling, ease of implementation, capturing full semantics of the application, and/or in handling advanced run-time dynamic requirements. As a matter of fact, various advanced features, such as objects, rules, roles, active capability and the flexibility of object-oriented database schema are needed in order to facilitate the development of a WFMS for handling NGIS [ChiuLi97], especially with respect to exception handling.

2. **Integrated Environment** - There are different frameworks for applying these advanced features to support NGIS such as loosely-coupled systems that extend and patch up existing databases inside the WFMS, and integrated environment that provides a coherent layer to support for the advanced features of the WFMS. The ADOME (ADvanced Object Modeling Environment) is an integrated environment developed to enhance the knowledge-level modeling capabilities of OODBMS models [LiLoc98], so as to allow them to more adequately accommodate the data and knowledge management requirements of NGIS. As such, it provides a suitable environment for developing, among other advanced information management applications, a versatile WFMS utilizing the offered facilities.
(3) Effective Coordination of Human Resources - A common cause for the failure of executing a task is either no agent is available or the wrong agent is assigned. Moreover, human resource is often the major cost in many organizations. Thus, effective coordination of human resources is a key to success of the WFMS and that of the organization.

ADOME-WFMS models a business process as a workflow (an activity) executed by a set of problem solving agents (PSA). In this paper, we use the terms activity and workflow interchangeably. A Problem Solving Agent (PSA) is a hardware/software system or a human being, with an ability to execute a finite set of tasks in an application domain. An activity is typically recursively decomposed into sub-activities and eventually down to the unit level called tasks (as illustrated in Figure 1). A task is usually handled by a single PSA. The execution of the activities is driven by the occurrences of events while the WFMS schedules and selects the PSAs for executing the tasks. We match the tasks with PSAs by using a capability-based token/role approach, where the main criterion is that the set of capability tokens of a chosen PSA should be match to the requirement of the task. A token embodies certain capabilities of a PSA to execute certain functions / procedures / tasks, e.g., programming, database-administration, Japanese-speaking, while a role represents a set of responsibilities, which usually correspond to a job-function in an organization, e.g., project-leader, project-member, programmer, analyst, etc. Each PSA can play a set of PSA-roles and hold a set of extra capabilities. For example, John is a Japanese analyst-programmer who is leading a small project; thus he may play all the above-mentioned roles (project-leader, project-member, programmer, analyst, etc.), and in addition holds an extra capability (token) of Japanese-speaking. Upon an exception, appropriate events will be raised so that the exception manager of ADOME-WFMS will take control of resolutions.

Thus, our ADOME-WFMS is characterized by using advanced modeling, an integrated environment, effective and efficient coordination of human resources and activities, and its facilities for exception handling. Our overall objective of effective support of exception handling is to reduce, as much as possible, human intervention and increasing reuse of activity and exception definitions.

In this paper, we concentrate on the taxonomy of different types of exception and their handling approaches in ADOME-WFMS and augment these with a novel solution based on workflow evolution. We believe that a sound taxonomy is an important basis for research and this presentation can summarized what we are doing. (Most other details in the system modeling for ADOME-WFMS were described in [Chiu+97]).

The rest of our paper is organized as follows. In section 2, an overview of exception handling by describing taxonomy of exceptions in WFMSs is presented. A spectrum of possible solutions to exception handling, ranging from manual and ad hoc ones to more radical and systematic ones based on workflow evolution are presented in Sections 3 and 4. An overview of the exception handling framework for ADOME-WFMS, detailing our general exception resolution approaches including a workflow evolution approach is presented in Section 5. Section 6 compares related work. Finally, we conclude the paper with our plans for further research in section 7.

2. Overview of Exception Handling in WFMS

Exceptions and failures are some of the basic issues to be addressed by contemporary information systems especially WFMSs, for which comprehensive treatment for exceptions and failures is crucial to their utility. However, currently there is no established framework for exception handling in WFMSs. Motivated by [EderLieb95], we classify exceptions into the following two dimensions:

1. Exception Source: external or workflow

2. **Exception Type**: expected or unexpected.

We further group exception handlers based on the following three aspects:

1. **Exception Handling Mode**: trivial, automatic, cooperative, manual, or failure

2. **Reexecution criteria**: optional, critical, repeatable or replaceable

3. **Exception Handler Type**: procedural or declarative.

### 2.1 Taxonomy of Exceptions

#### 2.1.1 Exception Source

This dimension classifies where an exception occurs:

- **External** exceptions arise from external components participating in the WFMS such as the operating system, DBMS, software applications, machines and equipment, etc. The internal mechanisms for these "black box" components are not known to the WFMS.

- **Workflow (internal)** exceptions are those relating to workflow management issues such as unable to find a PSA or to get some resources required for executing a task, missing deadline or over running budget, special-case outcomes from a finished task, etc. We may further distinguish workflow exceptions into *inter-workflow* exceptions relating to 2 or more activities (such as resources competition) and *intra-workflow* exceptions that occurs within a workflow instance, though in this paper our focus is on the latter.

#### 2.1.2 Exception Type

This dimension classifies the knowledge of the WFMS on a particular occurrence of an exception:

- **Expected** exceptions are those anticipated and already planned with explicit exception handlers, which may have been supplied by the WFMS, the workflow administrator, or the user.

- **Unexpected** exceptions on the contrary require human intervention since it is unanticipated. Administrative rules can specify who should be informed to take action, though the solution itself is not in the WFMS. The WFMS can then provide suggestions for the human expert to plan and select alternatives to overcome unexpected exceptions, otherwise they will cause failure of the parent activity. The human expert can also choose to save this solution into a database by amending existing activity definitions so that this exception becomes expected and can be handled automatically if it occurs again.

### 2.2 Taxonomy of Exception Handlers

#### 2.2.1 Exception Handling Mode

This dimension represents the difficulty of handling an exception with respect to the knowledge of WFMS available and the effort required:

- **Trivial** - the exception (usually external and expected) is so basic that it is handled silently by the relevant system components and the WFMS does not even know its occurrence.

- **Automatic (system driven)** - the WFMS attempts to handle the exception (with an optional message to the target agents) automatically by finding an explicit exception handler or by pre-specified resolution. If the exception is not resolved, the target agents will be requested for action.

- **Cooperative (system assisted)** - a human expert is informed of the exception and request for instruction or approval so that the WFMS can carry out subsequent exception handling tasks.

- **Manual (user driven)** - a human expert is requested for exception handling and the progress of this activity is put on hold until the expert fixes the problem.
• **Failures** - the exception can be handled neither by the WFMS nor any available human experts and thus causing an exception to the parent activity initiating this task.

By default, automatic handling is attempted. However, if an exception handler cannot be determined, the WFMS will request for human invention so that cooperative or manual handling can be carried out. If that fails too, the activity fails completely. To ensure proper degree of human intervention, the WFMS should allow users to override the intrinsic exception handling mode for important tasks and activities in the WFMS. For example, even if an exception type is well expected (e.g., to hire more programmers if coding fails to meet the deadline) but we specify manual handling for critical tasks (e.g., a multi-million dollar software project), the WFMS should still inform the top management and seek for manual intervention.

### 2.2.2 Re-execution criteria

Users can specify the following re-execution criteria (or re-execution pattern) for different tasks and sub-activities to assist in automatic exception resolution:

- **Optional** - whether the task is successful will not affect the next step of the transaction or the success of the parent activity.
- **Critical** - once the execution path is selected but the task fails, it cannot be repeated nor can alternate solutions be determined. Unless there is human intervention, it causes failure of the parent activity.
- **Repeatable** - once the execution path is selected but the task fails, it can be re-executed by the same agent, alternate agents, or using alternate resources; but the WFMS should not try other alternate paths because of some reasons. (E.g. due to set up overhead for a production line or a process etc.)
- **Replaceable** - should the task fail, the WFMS can choose either redo the task with free choice of agents and resources; or if necessary, the WFMS can choose any available alternate execution path to work around with the problematic tasks.

### 2.2.3 Exception Handler Type

Users can specify the following re-execution criteria (or re-execution pattern) for different tasks and sub-activities to assist in automatic exception resolution:

- **Procedural** - These are extra branches of activity decomposition for exception handling. Each procedural handler is specific to a particular context for handling specific outcomes.
- **Declarative** - These are ECA rules specifying resolution actions under certain events and conditions.

### 3. Exception-handling Resolution Techniques

From an action point of view, different levels of resolution are possible during manual or automatic exception handling, which includes.

**Level 1. Resolution by maintaining execution behavior**

**Level 2. Resolution by modifying execution behavior**

**Level 3. Resolution by evolving workflow**

We shall discuss Level 1 and Level 2 in this section, which are case-by-case resolutions selected by users manually upon each unexpected exception; Level 3 is discussed in the next section, which aims at avoiding exceptions and/or subsequent automatic exception handling. The detailed resolutions to be described below are those supported by ADOME-WFMS.

### 3.1 Resolution by Maintaining Execution Behavior

Updating data objects like PSAs, resources, etc., and changing activity constraints (e.g., budgets and/or deadlines) are the
simplest and most fundamental solutions for handling basic workflow exceptions such as inadequate PSA availability or activity constraint violations. The user or management does not need to understand every detail of the workflow, but probably only the specific task causing the exception in order to make a sensible decision for the exception. Possible resolution decisions include the following:

**Changing capability or resource requirement for a task instance** - under exceptional cases when there are no suitable PSA or adequate resources, the management may decide to temporarily relax the requirements for a task instance. Thus, the WFMS can automatically (or the management can manually) select PSA with less capability to carry out a task. For example, because a certain job is small, a PC instead of a workstation can be used; a junior programmer, instead of a senior programmer, may take full responsibility of the job. On the other hand, the management may decide to increase the capability requirement upon failure of a task when the cause is due to inadequate experience or capability. For example, because a certain job is very complicated and the chosen programmer fails to finish the job, the management may request a senior programmer to redo the job.

**Changing constraints for an activity** - an activity or a sub-activity usually have various global constraints such as deadlines or budgets. There may be other constraints imposed such as total number of PSAs allowed for a particular project (instance) or a whole sub-activity (class) must be carried out by a single PSA. Many of these constraints may probably reflect preferences rather than absolute critical requirements. The management would judge and probably prefer relaxing such constraints instead of a failure for a certain activity (which means loss in work done), especially if there are no other cost-effective alternatives.

**Amendment of capabilities for one or more PSAs** - through processes like training of personnel, accumulation of experiences, and upgrading of machines, the capabilities of PSAs can increase over time. For example, an organization may decide to train their LAN administrators so that they can set up and manage their new web site instead of hiring additional staff. On the other hand, incomplete specification of capabilities in the system is often possible, and thus requires amendment in view of exceptions to occur. For example, upon the event that the art designer needs a lot of assistants in a major web-authoring project, volunteer colleagues with art creativity will be called for and thus the capability "art creativity" will be added for those suitable volunteers. (Without this event, such assessment would most probably not be able to carry out.)

**Adding PSA or other resources** - if all the PSAs which can carry out a task are occupied or all of them are not available, the management may consider adding more PSAs of the required type, especially if the organization is often in short of such type of PSA. Consider for example, when there is a shortage of programmers but there are many new projects, a simple solution is to hire more programmers. Moreover, if there are actually no PSAs in the organization capable of doing the task, adding a suitable PSA is a natural solution. For instance, if an organization needs to set up and maintain a comprehensive web site of its own, they may want to hire a new web administrator. In some cases, adding PSA may speed up the tasks (e.g., hiring more programmers for coding), and thus help in meeting deadlines. The consideration factors for acquiring additional resources are similar to that of handling inadequate PSAs.

### 3.2 Resolution by Modifying Execution Behavior

Upon exceptions, the management may manually specify alternative and/or additional steps of a workflow instance instead of the regular flow so as to solve an encountered problem. However, this requires deeper understanding of the semantics of the affected workflow (at least the affected sub-activity). These include:

**Waiting** - hold the progress of an activity and wait for some other events before resuming execution. Typically, this occurs when the required PSAs are unavailable or there are inadequate resources but the task is not so urgent and thus can afford waiting. The management may also specify waiting before re-execution of the failed task or an alternate solution so that the environment may change in favor of the task. (E.g., wait for the price of a raw material to get back to normal before purchasing them.)

**Skipping current task** - this corresponds to the manual decision for the "optional" execution criterion. The management may decide whether the skipped tasks should be re-executed later or just ignored. In any case, the next step of the workflow will be executed since this optional task does not affect the progress.

**Repeating current task** - this corresponds to manual decision for the "repeatable" execution criterion. The management may choose the same PSA, alternate PSA, or use alternate resources to repeat a failed task. Choosing to repeat a task instead of other alternatives often helps reduce the expense of preparation and/or set up work.

**Switching PSA assignment for tasks** - if a new complicated task requires an expert PSA with many capabilities but some
of those PSAs are engaged in simpler tasks that actually do not require all the capabilities. In this case, an expert PSA may be swapped out of his assigned task to perform the new complicated task which requires his expertise, letting another PSA with less but enough capability to take over the simpler task. For example, suppose the programming team has only 2 programmers: John can program in Prolog and C but Tom only knows C. Let John be engaged in project t1 but Tom is free. Now a new project t2 needs a Prolog programmer and thus Tom replaces John in the old project so that John can contribute to the new project. Figure 2 intuitive illustrates such a desired PSA assignment swapping dynamically.

Choosing an existing alternate branch for execution - if the required PSA or resources are not available for execution of a task, the management may choose other feasible alternate branches (if any) instead. This will not cause loss in work but may result in a higher cost or a less satisfactory result. For example, although it is better to hire an in-house administrator to set up and maintain a comprehensive web site, it is still feasible to accomplish the task by passing the job to an external web-hosting service company. Similarly, if a suitable web-hosting company cannot be found for setting up some web pages and instead a web administrator is hired, a higher cost may be incurred. On the other hand, some tasks may fail due to wrong initial judgement of feasibility or due to changes in environment and thus they should not be repeated. The correct solution should be attempting another branch for execution in such cases.

Aborting the sub-activity - under situations where a task fails and there are no feasible alternatives, or there is a severe shortage of resources or agents, or the sub-activity is seriously over budget or beyond deadline, the management may choose to abort the execution of the whole sub-activity and seek solution at a higher level activity. The extreme case is the whole activity or project is aborted (e.g., when aborting the project can stop further loss of money).

Aborting other tasks - the management may decide to abort other less important tasks so as to release resources or PSAs for more urgent and important ones.

4. Workflow Evolution

While a conventional approach to exception handling is often manual, off-line and on a case by case basis, a more radical approach would be to allow at run-time various kinds of changes of workflow to be specified and accommodated, i.e., modification of workflow definitions during work progress. With this approach, more exceptions could be avoided and/or handled automatically should they occur repeatedly. This reflects the accumulation of experience and knowledge by the WFMS, which may lead to better solutions and/or migration of procedural knowledge to declarative knowledge [MarWoo92]. In this section, we describe the different categories of workflow evolution useful and relevant to exception handling.

4.1 Schema Evolution

Schema evolution can be supported only if the underlying OODBMS has such functionality; examples include the following:

Changing capability or resource requirement for a task (class) - instead of temporarily changing the capability or resources requirement for an instance, after getting enough experience and observations, the management may decide to permanently change them at the schema level so as to avoid further exceptions of this kind. This may be considered some form of migration of experience to declarative knowledge [MarWoo92]. Moreover, the management may decide to change these requirements for other reasons such as cost, quality, and/or speed.

Changing capability token / role hierarchy - this is necessary due to incomplete specifications and unanticipated requirement changes. Composition links and/or Isa links may sometimes be missed during the initial specification and
thus may result in failures in finding a suitable PSA (although one might be actually available). Upon failure of tasks, the management may deduce that a new kind of capability is crucial to the success of a task. Thus, the new capability definition should be added to the WFMS, and the task requirement should also be updated, along with other details (such as which PSAs possess this new capability) to be found out and specified.

**Changing organizational structures** - The management may exercise this type of functions upon encountering exceptions due to shortage of resource or PSA since eliminating redundant structures and combining small project teams or units may sometimes result in more efficient use of resources.

### 4.2 Patching workflow definition to avoid exceptions

With more thorough understanding of problems after one or more occurrences of an exception, the management may discover the actual reason for the exception. If the exception can be avoided, it is better to avoid it rather than let it occur from time to time because of the overhead of exception handling. This is especially useful if there are more than one possible branches of workflow. Typical modifications include:

**Changing pre-conditions for transitions** - Extra checking and/or pre-requisites are added before the execution of a branch of the workflow. For example, to avoid shortage of Prolog programmers during the project, a condition check of "more than 3 Prolog programmers available in team" is added prior to the transition to the task of "selecting Prolog as the programming language" for a project.

**Eliminating problematic branches and adding new branches** - some unreliable procedures may be totally removed to avoid exceptions, if there are more reliable alternatives (though may incur higher costs). New alternatives can be added and less reliable alternatives may be downgraded to serve only as back-ups or simply be removed. For example, because external services are often late, the management may decide to hire more programmers and not to contract out small software development jobs.

**Extra preparation work** - extra preparation work may improve the working environment and eliminate adverse conditions leading to exceptions. For example, after a severe data loss due to computer virus attack, all systems delivered to clients now include pertinent virus checking software and procedures.

### 4.3 Explicit procedural exception handler

With experiences gathered from the manual handling of exceptions, it may be possible for the management to later specify explicit procedural exception handlers so as to perform automatic exception handling. In this way, unexpected exceptions now become expected ones and the system can be more versatile and more efficient; possible procedural handlers may include:

**Procedural exception handlers before main task** - the procedural handlers are quite similar to preparation work but are only executed upon specified exception condition. For example, if no vehicles are available for delivering the goods, call and hire one. Moreover, actions like changing PSA assignment and aborting other tasks to release required PSAs/resources can be added, along with suitable conditions guiding such executions.

**Re-execution pattern** - these include changes in execution behavior such as skipping or repeating the current task, choosing another existing branch for execution, or aborting the current sub-activity (as discussed in section 3.2). Once such actions are stabilized or become permanent, subsequent exceptions can follow the same way by re-executing these actions and thus no more human intervention will be required.

**Compensation or additional action** - upon failure of some tasks, some compensation activity and/or additional procedure may be required to execution. For example, if a file containing the source code of a program was deleted from the hard disk, the programmer need to load it back from a backup tape before he can continue to work on it.

### 4.4 Declarative ECA rules for exception handling

Upon different situations (events and conditions) which may cause directly or indirectly the exceptions (or may be unrelated to exceptions at all), declarative ECA rules are useful for specifying generic actions in bulk, thereby enhancing the modeling power of the WFMS significantly. For exception handling specifically, ECA rules are useful for:
Handling violations of global constraints - this represents a more specialized use of ECA rules, where exceptions relating to violations of a global constraint may probably be handled only within a specific workflow context (e.g., to postpone deadline, adding budgets, etc.).

Specifying summarized rules for exception handling such as re-execution patterns, criteria for adding PSA and resources, aborting sub-activities, PSA reassignment, etc.

It should be emphasized that ECA rules can be at fairly high level and can cover different dimensions of objects, including:

- Association of actions with a class of tasks / sub-activities occurring in any context or in a specified context upon exception. This allows exception handlers not to be tediously specified individually for each task of every activity.
- Association of certain PSA classes with actions. E.g. all tasks of trainees should be logged, all exceptions relating to trainees should be reported, etc.
- Association of actions with certain events in general. E.g. upon a hard disk failure, report to the Information Systems manager.

4.5 Other Resolutions

Other possible resolutions include drastic amendment to structures of the task and sub-activities at run-time. The minimal sub-activity containing all modifications can be restarted or resumed at a user-specified point due to the encapsulating properties of the composition hierarchy of activities and tasks. Changes in data / objects or re-executions are relatively simple solutions which can be supported easily. Such a change can be effected either manually or by automatic exception handlers. However, schema and workflow evolution usually must be proceeded with human intervention.

In next section, we describe how ADOME-WFMS accommodates this entire spectrum of exception handling resolutions through its flexible architecture in a unified manner.

5. Exception Handling Framework for ADOME-WFMS

5.1 Exception Handlers in ADOME-WFMS and Scoping

In ADOME-WFMS, conceptually exception handlers can be:

- **Procedural**: There are extra branches are for exception handling. Each procedural handler is specific to a certain task or sub-activity under a particular context for handling specific outcomes.
- **Declarative**: Since activities are organized in a composition hierarchy, declarative exception handlers in the form of ECA rules can be specified within the scope of different activity and sub-activity levels. Thus, an exception handler applies not only to the body of the target activity but also to all sub-activities and tasks. For example, some tasks within a sub-activity are expected to raise a specific exception type (e.g., program_error), an exception-handling ECA rule with a matching event type in the 'E' part can handle the exception accordingly. (E.g., E: program_error; A: inform(programmer))

Both kinds of exception handlers can be added, deleted and modified during activity definition time before execution, or during exception occurrence at run-time (viz. workflow evolution), which are adequately supported by the dynamic schema evolution capability of ADOME.

When an exception occurs, the search for a handler is from the current task to its parent, then progressively up to the global activity (unless it is stopped by an explicit declaration). This also allows for special exception handlers overriding default exception handlers if necessary. For example, a declarative exception handler (rule) can specify that all exceptions should notify the personnel manager in the recruitment activity while a global exception handler specifies all PC failures should be reported to the EDP department. In this case, the failure of the PC in the personnel department causing exception to a sub-activity of the recruitment activity will trigger both rules to inform the personnel manager and the EDP department. However, the user can specify in some non-critical sub-activities / tasks to inform, e.g., a personnel
officer on duty, instead of the personnel manager as an improvement of the exception handler.

If an exception is not handled (either due to the WFMS or expert's decision) in a sub-activity, the activity fails and triggers exception to its parent activity. This process may propagate up the composition hierarchy until it is handled. This approach localizes exception, and thus reduces loss of work done.

Similarly, human intervention requirements of exception handling (automatic, warning, semi-automatic and manual) and re-execution patterns (optional, critical, repeatable and replaceable) for sub-activities and tasks are specified within the scope of this composition hierarchy, with the lowest level taking priority in specification and thus overriding those of higher levels.

5.2 Reusing Exception Handlers

Since exceptions can be rather common in a WFMS, reusing exception handlers is vital to the effectiveness, user-friendliness and efficiency of the WFMS. In ADOME-WFMS, mechanisms for reuse of exception handlers follow from its structure:

- For procedural exception handlers, arcs from more than one peer tasks / sub-activities of the same level (siblings inside the same parent activity) can lead to the same exception handler for some degree of sharing.
- Because of scoping, only one declarative exception handler is required for each exception type for a whole activity composition hierarchy (as explained in the previous sub-section).
- Declarative exception handlers are first-class ECA rule objects. A rule object \( r \) is required to be declared and defined once and then can be associated with more than one scope by repeated binding. (E.g. Bind \( r \) to activity1, activity2).
- Since exceptions are events (which are first-class objects in ADOME), exception classes are also arranged into an 'isa' hierarchy. Thus, an exception handler for a super-class will also handle an exception of a sub-class. (E.g., an exception handler for program_error will handle subscript_out_of_range also.)
- Extending the event-part with 'or' event composition can generalize exception handlers. (e.g., E: program_error / computer_breakdown, A: inform(EDP))

5.3 ADOME-WFMS Exception Manager

Internally, the activity decomposition module rewrites user-supplied ECA rules and error-handling procedure definitions so that when an exception occurs, an event is triggered to first invoke the exception manager. The exception manager then takes control and acts according to different exception handling mode discussed in Section 2.1. Relevant users will be notified unless automatic handling mode is specified. Figure 5 depicts the control flow of the ADOME-WFMS exception manager.

![Figure 5: Meta-activity for ADOME-WFMS Exception Manager](http://ccs.mit.edu/klein/cscw98/paper16/)

Note that if human intervention is required, the human intervention manager will be invoked, thereby various exception-handling activities can be specified and executed (and the user can request for automatic resolution too). If there is an explicit, a resolved, or manually specified handler, then the chosen handler will be executed; otherwise human intervention will (again) be called for. In some other cases, the exception handler or the human may abort the parent (sub)-activity if it is determined that a failure has occurred and the parent sub-activity cannot be continued.
5.4 Supporting Exception Handling in ADOME-WFMS

In Section 2, we have described different types of exceptions and here we describe how ADOME-WFMS can handle them.

5.4.1 Expected External Exceptions (EEE)

There can be several kinds of expected external exceptions where usually declarative ECA handlers are related with actions for them:

- Some of these exceptions have been handled by the corresponding system level components participating in the WFMS, such as DBMS, operating systems, network. Well-known techniques (e.g., rollback, roll-forward for DBMS) are applicable to handle these failures.
- Some of the above exceptions are handled silently by the external agents and ADOME-WFMS does not even need to be aware of (trivially handled), while others can notify the ADOME-WFMS by raising an appropriate event so that further actions can be taken (e.g., for logging and audit trail).
- Other exceptions, though expected, cannot be handled by the external system components. Appropriate events are raised to inform ADOME-WFMS so those explicit exception handlers (which have been specified by the workflow administrator or users) can be executed.

5.4.2 Expected Workflow Exceptions (EWE)

For certain types of exceptions, which are quite general, ADOME-WFMS has built-in exception handlers for them. For example:

- If a PSA rejects a task assignment or the best candidate PSA is not available, the WFMS will find the next available PSA.
- If all PSAs capable of executing the task are busy or the required resources are occupied, the WFMS will either wait or choose alternate execution paths.

For more specific exceptions, the workflow administrator or users will have to anticipate and input exception handlers so that the WFMS can handle these workflow exceptions for them.

ADOME-WFMS supports a lot of exception handling resolutions relating to PSA assignment based on capability matching, such as amending of capabilities for PSAs and changing capability requirement for a task instance. This is important because a significant proportion of internal (workflow) exceptions are due to failures in finding (suitable) PSA(s) for the execution of tasks. Moreover, ADOME supports advanced analysis for PSA capabilities termed as "capability role/token multiple inheritance hierarchy" and "token derivation network theory" [Chiu+97]. This greatly helps increase the chance of finding suitable PSA(s) automatically (thus avoiding no-PSA exceptions due to misunderstandings of PSA capabilities), and also finding alternate ones for repeating a task upon exception. For automatic switching PSA assignment among tasks, these advanced capability processing features are even more vital to the success of this resolution scheme. It should be noted that quite a number of traditional WFMS like Flowmark [Alonso+94] and OASIS [MarWoo92], do not readily support or employ the notion of capability matching for PSA assignment to tasks.

ADOME-WFMS supports different re-execution criteria / patterns so that failed tasks may be re-executed or alternate task may be executed instead. Moreover, ADOME-WFMS can resolve and decide for the correct alternate PSAs or alternate execution branch automatically if the re-execution pattern for a task has been specified. To our best knowledge, very few other WFMSs (except WAMO [EderLieb95]) support these features.

5.4.3 Conventional Approaches for Unexpected Exceptions

For an unexpected exception, the treatment of workflow and external exceptions is similar. Upon unexpected exceptions, the human intervention manager sub-module of the Exception Manager will assist the user by providing a list of possible resolutions with some evaluations. Moreover, all recent case by case resolutions are kept in the database for user reference. Since every scenario can be different, only the user can probably determine what is the most appropriate action. (A comprehensive system assisting such decision-making is beyond our scope.) The following table summarizes the suggested resolution for some exception situations from a situation point of view.

http://ccs.mit.edu/klein/cscw98/paper16/
In short, ADOME-WFMS provides an adequate support of a variety of conventional exception resolutions as studied in Section 3, not only manually but also automatically for many of them. For example, the resolution of abort in the current sub-activity or other tasks / sub-activities is well supported, since such an exception can be handled in a unified manner at a higher level (the parent activity of the activity composition hierarchy) either automatically or manually. In addition, ADOME-WFMS also supports radical solutions based on workflow evolution as discussed below.

5.4.4 Workflow Evolution for Unexpected Exceptions

ADOME-WFMS has the required facilities for supporting various types of workflow evolution (cf. Section 4). In particular, besides conventional exception handling resolutions, the human intervention manager sub-module also accepts update of workflow on-line. In contrast, there are currently few WFMSs having such facilities for supporting the whole spectrum of exception-handling resolutions, especially those relating to workflow evolution. In ADOME-WFMS, the user can choose any suggested resolutions to be permanent ones or enter any schema evolution operation, update of workflow and/or enter new ECA rules. As workflow evolution requires the modification of workflow definitions or adding ECA rules to the system during work in progress, advanced schema evolution capability is required at run-time. Many WFMSs based on relational databases, such as CapBasED-AMS [Karlap+95], can hardly support schema evolution and thus severely restricts workflow evolution resolutions. ADOME-WFMS readily provides exception resolutions based on schema evolution (cf. Section 4.1) due to ADOME's support of dynamic schema evolution [LiLoc97]. It should be noted that the resolutions based on schema evolution are general-purpose ones, which can help to reduce further exceptions to occur. Furthermore, advanced ECA rule support in ADOME [ChanLi96,ChanLi97] greatly facilitates the reuse of exception-handling rules and the flexibility of associating these rules for different targets such as task / sub-activities, PSAs and events (cf. Sections 4.3 and 5.3.2).

Because ADOME-WFMS uses activity decomposition, upon workflow evolution (i.e., modification of a certain sub-activity class definition), the side effects of affecting other activities containing this sub-activity is very much confined. At the time of the workflow evolution, only those activities having the same sub-activity in execution are affected. Other activities having the same sub-activity but not in execution are unaffected since the sub-activity is encapsulated and behaves as a black box to activities at a higher level.

5.5 Illustration of Exception Handling in ADOME-WFMS

Throughout the previous parts of this paper, we have been using examples from a software development domain. As another more complete example for more conventional workflow applications, Figure 6 captures the requisition procedures of a newly formed company. Concentrating on the specification of the normal cases, the only procedural exception handling specified is 'supplier not found' (cf. Figure 6(a)), which is quite clear to the administrator.

The next step is to specify, with scoping in ADOME-WFMS, re-execution patterns for activities and tasks so as to save many tedious explicit jumps and aborts. More specifically:

- The 'Requisition' activity specified in Figure 6(a) is repeatable and thus 'purchase request', 'procurement' etc. are
all repeatable unless otherwise stated.

- As funding may not be available for cash on delivery (COD) or the supplier may not be willing to deliver the ordered goods if the new company's credit limit is exceeded, the two branches following 'Procurement' (cf. Figure 6(a)) are replaceable.

- Upon 'Purchase Request', the user may not need to 'get product information' because he may know that very well or he does not even know how to get such information. (E.g., the user may just specify that he wants a chair of around $500 and let the procurement department take care of the rest.) Hence this task is optional (cf. Figure 6(b)).

- As illustrated in Figure 6(b), tasks 'Budget check' and 'PR approval' are flagged critical so that the sub-activity 'Purchase Request' will fail immediately if these tasks cannot be passed. However 'Purchase Request' is repeatable so that the user can also revise the budget and/or product specification so as to retry for approval.

To cater for special cases, declarative exception handlers can be specified. For example:

- If financial controller is not available, the department manager can approve purchase request of less than $10,000 in the 'PR approval' task.

- Send Email to supplier if invoice is lost in the step 'Match PR, PO and invoice'

- All exceptions in the 'Payment arrangement' activity are regarded as to having severe consequences, so they will trigger E-mail to the financial controller; and for the same reason, manual handling of exception is required.

After running the WFMS for some time, the administrator may decide to change the 'Procurement' activity (cf. Figure 6(d)) to version 2 (as detailed in Figure 6(e)) by adding additional arcs and tasks for exception handling. Also, if many suppliers complain late payment and refuse to deliver ordered goods, the 'credit' branch of Figure 6(a) may be completely deleted. Thus, instead of delaying the issue of cheques for credit payment after receiving the goods, the new arrangement may be that a post-dated cheque is issued to the supplier upon each delivery of goods. (The difference between credit and COD payment in this case will be just the date on the cheque and the payment procedure becomes much simplified.) These are some examples of workflow evolution in ADOME-WFMS, which can effectively support handling those more complicated exceptions that are difficult (if no impossible) to handle using the conventional resolutions. As a by-product, such a workflow evolution approach also helps lead toward (eventual) exception avoidance.

6. Related Work

There have been some notable efforts in developing advanced WFMSs offering various interesting features. Among them, TriGSflow [Kappel+95] is perhaps the closest system to ours in that it also adopts an OO design, and utilizes features like rules and roles. However, it only uses roles for associating PSAs with tasks, but not for modeling capabilities needed in matching agents with tasks. Moreover, it has little support for handling exceptions.

[Kumar+96] also described a framework of using roles, event-based execution model and exception handling rules in a WFMS based on document flow. However, they do not provide a comprehensive organization model nor address a variety of exception conditions. Moreover, they do not support activity decomposition or capability matching to facilitate workflow definition and execution.

![Workflow Diagram]

http://cs.cmu.edu/~klein/escw98/paper16/
### Class Requisition is Activity

- **Sub-activities**: purchase_request, procurement, payment_arrangement, receive_check_goods, wait_payment_due;
- **input_events**: requisition_start;
- **output_events**: requisition_end;
- **input_parameter**: purchase_list (set of resources);
- **output_parameter**: expenditure (real);
- **reexecution_pattern**: repeatable;
- **Activity_type**: user_defined;
- **Rules**
  - R1: E: end_of_task, C: pr.budget > 1000000, A: log(task); inform(task, CEO)

### Class Procurement is Activity

- **Sub-activities**: find_suppliers, get_product_info, prepare_po, user_evaluation, purchase_approval, supplier_confirm;
- **input_events**: procurement_start;
- **output_events**: cod, credit, supplier_not_found;
- **input_parameter**: pr (purchase_request_document);
- **output_parameter**: po (purchase_order_document);
- **reexecution_pattern**: repeatable;
- **Activity_type**: user_defined;
- **Rules**
  - Activity_type = manual;

### Class get_product_info is task

- **Task_need**: procurement_clerk;
- **input_events**: get_product_info_start;
- **output_events**: user_evaluation_start, budget_too_low;
- **input_parameter**: pr (purchase_request_document), s (set of supplier);
- **output_parameter**: q (set of quotation);
- **Activity_type**: manual;

### class revise_budget_request is task

- **task_need**: manager;
- **input_events**: budget_too_low;
- **output_events**: user_evaluation_start;
- **input_parameter**: pr (purchase_request_document), q (set of quotation);
- **output_parameter**: pr (purchase_request_document);
- **reexecution_pattern**: critical;
- **Activity_type**: manual;
WIDE [Casati+96] also uses contemporary modeling techniques and has basic measures in handling exceptions. However, systematic analysis and classification of exceptions and their handling approaches are not addressed.

WAMO [EderLieb95] uses the concepts of Sagas and flexible transactions for supporting workflow exception handling. It also offers a preliminary classification of exceptions upon which we had made some necessary extension in our taxonomy of exception categories (cf. Section 2.1).

OASIS [MarWoo92] models an organization as a network of MOAP (micro-organization) nodes and provides a model for organizing and structuring organizational knowledge. Thus, its structure is different from the above-mentioned systems which all use a composition hierarchy of tasks. OASIS uses classes and objects concept to assign the task to individual MOAPs.

Flowmark [Alonso+94] also uses Sagas and flexible transactions directly for modeling workflow exception handling. To avoid system failures, Flowmark even uses replicated database and clustered workflow servers. Exotica/FMDC [Alonso+95] extension of Flowmark also handles disconnected agents. These additional techniques should be relevant in handling exceptions since managing manual tasks off the computer system (e.g., meeting, typing, etc.) are not much different from handling disconnected agents in nature. Since Flowmark only finds out all possible candidates for a task and then lets them volunteer for the execution instead of using capability matching.

---

**Figure 6 - Example Workflow of requisition procedures**

```
Rules

EDPrule= (E: purchase_approval_start || revise_budget_request,
      C: intersect(pr.item, computer_resources) <> empty_set,
      A: inform(task, EDP))
End;

Class find_supplier isa task
  task_need = (procurement_clerk);
  input_events = (procurement_start, user_reject_suppliers,
                management_reject_supplier);
  output_events = (get_product_info_start, supplier_not_found);
  input_parameter = (pr: purchase_request_document);
  output_parameter = (s: set of suppliers);
  Activity_type = manual;
  ...
end;

class user_evaluation isa task
  task_need = (employee);
  input_events = (user_evaluation_start);
  output_events = (purchase_approval_start, user_reject_suppliers);
  input_parameter = (pr: purchase_request_document,
                      q: set of quotation);
  output_parameter = (chosen_supplier: supplier);
  Activity_type = manual;
  ...
end;
```

---

effectiveness and fairness may be impaired as a consequence.

ConTract [ReuSchw95] focuses on activity control and execution but not on organization modeling and activity specification and decomposition. ConTract net protocol is for communication among nodes in a distributed problem-solving environment, where nodes engage each other in discussions that resemble contract negotiation for PSA / task assignment. Similarly, other work like [Geoga+96] focused on transactional aspects and is thus at a lower level of concern.

CapBasED-AMS [Karlap+95] is based on a relational system with added active capability. The modeling for workflow is not straightforward since almost all designs (which are in essence object-oriented) need to be converted to traditional relational model. Moreover, the system can hardly provide support for actions requiring advanced OODBMS features, such as dynamic schema modification (basic requirement for workflow evolution) at run-time.

In summary, other workflow systems either do not address exception problems comprehensively or concentrate only on extended transaction models.

7. Conclusions

This paper has described a framework of exception handling in ADOME-WFMS, a flexible WFMS based on ADOME which is an active OODBMS extended with role and rule facilities. In the context of ADOME-WFMS, we have presented a taxonomy of different types of exception, along with a comprehensive list of the possible handling approaches and their specific exception handlers. A notable solution supported by ADOME-WFMS to handle more complicated exceptions is based on workflow evolution - an approach seldom adopted by other WFMSs. ADOME-WFMS is currently being constructed upon the ADOME system.

Further research work currently being carried out includes: cost model for effective PSA allocation and alternatives in exception handling; issues in cascaded and nested exceptions handling, maintenance of previous cases of human intervention for future reference and automatic case-based exception handling; a more comprehensive methodology of handling exceptions systematically; and enhancement of the prototype system with the targeted exception handling features and workflow evolution support.

References


[ChanLi97], L.C.Chan, Q.Li, "An Extensible Approach to Reactive Processing in an Advanced Object Modeling


