A Logical Framework for Exception Handling in ADOME Workflow Management System♣

Dickson Chiu[1], Qing Li[2], and Kamalakar Karlapalem[1]

[1]Department of Computer Science, University of Science and Technology,
Clear Water Bay, Kowloon, Hong Kong
{kwchiu,kamal}@cs.ust.hk
[2]Department of Computer Science, City University of Hong Kong,
Tat Chee Avenue, Kowloon, Hong Kong
csqli@cs.cityu.edu.hk

Abstract. We have been developing ADOME-WFMS as a comprehensive framework in which the problem of workflow exception handling can be adequately addressed. In this paper, we present detailed design for ADOME-WFMS with procedures for supporting the following: an integrated approach starting from exception detection to exception resolution, reuse of exception handlers, and automated resolution of expected exceptions.

1 Introduction

Exception handling in workflow management systems (WFMSs) is a very important problem since it is not possible to specify all possible outcomes and alternatives. Effective reuse of existing exception handlers can greatly help in dealing with workflow exceptions. On the other hand, support for workflow evolution at run-time is vital for an adaptive WFMS. Any WFMS that provides a comprehensive solution for exception handling needs to provide a framework for the specification and support for generic exception handlers that can be fine-tuned to handle different run-time exceptions. Reuse of workflow definitions and exception handlers are very important in the smooth operation of a flexible WFMS.

In this paper, we model a business process as a workflow (an activity) executed by a set of problem solving agents. 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. Typically an activity is recursively decomposed into *sub-activities* and eventually down to the unit level called *tasks* (as illustrated by the example in Fig. 3). A task is usually handled by a *single* PSA.The WFMS schedules and selects the PSAs for executing the tasks. We match the tasks with PSAs by using a capability-based *token/role*

♣ This research has been partly funded by HKSAR RGC Grant HKUST 747/96E.

B. Wangler, L. Bergman (Eds.): CAiSE 2000, LNCS 1789, pp. 110-125, 2000

approach [19], where the main criterion is that the set of capability tokens of a chosen PSA should be matched 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.

In this context, we employ an Advanced Object Modeling Environment (ADOME [23]) to not only specify generic exception handlers but also facilitate the reuse and adaptation to handle specific instances of exception that occur at run time. In particular, our ADOME exception handling environment facility provides the following features:

- Dynamic binding of exception handlers to classes, objects and roles, and to activities with scoping.
- Addition, deletion and modification of handlers at run-time through workflow evolution support.
- Specifying and reusing exception handlers upon unexpected exceptions with the Human Intervention Manager.

In contrast with traditional software systems, workflows usually evolve more frequently, making reuse a vital issue. On the other hand, workflow evolution often takes place at execution time, making it much more difficult to handle. There have been few WFMSs designed to address these two problems (viz. reusing exception handlers and workflow evolution) effectively and adequately. In this regard, we have been developing ADOME-WFMS as a comprehensive framework in which the problem of workflow exception handling can be adequately addressed.

We use an integrated, event-driven approach for execution, coordination, and exception handling in our WFMS. Events (such as database events / exceptions, or external inputs) trigger the WFMS *Activity Executor* to start an activity. The WFMS *Activity Executor* uses events to trigger execution of tasks, while finished tasks will inform the *Activity Executor* with events. Upon an exception, exception events will trigger the WFMS *Exception Manager* to take control of resolutions.

In this paper, we present a logical framework of exception handling (with detailed design and main algorithms) for ADOME-WFMS, covering reuse issues in specifying exceptions and their handlers and a novel solution based on workflow evolution. More details regarding classification of exceptions and handlers, and modeling aspects for ADOME-WFMS are given in [8] while ADOME-WFMS exception driven workflow recovery are presented in [9]. The objectives and contribution of this paper include: (i) the mechanism of the ADOME-WFMS and resolution of expected exceptions, (ii) support of reuse for workflow definitions, constraints, exception types and handlers in ADOME-WFMS, and, (iii) demonstration of the feasibility of ADOME-WFMS for effective support of exception handling through effective reuse.

The rest of our paper is organized as follows. Section 2 describes the architecture and basic execution mechanisms of ADOME-WFMS. Section 3 presents how

ADOME-WFMS resolves for expected exceptions. Section 4 explains how reuse can be facilitated. Section 5 compares related work. Finally, we conclude the paper with our plans for further research in Section 6.

2 Architecture of ADOME-WFMS

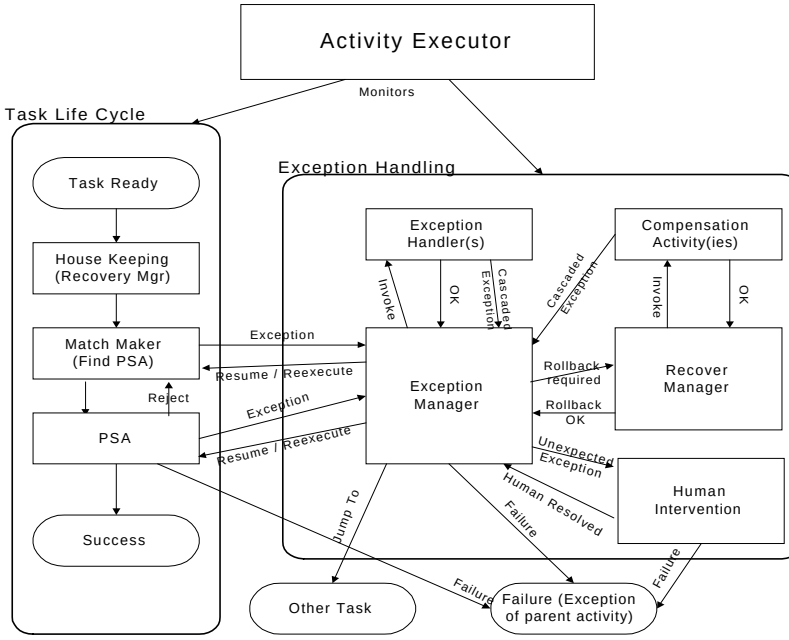

Fig. 1. ADOME-WFMS Task Execution, Exception and Recovery

The ADOME system was developed to enhance the knowledge-level modeling capabilities of OODBMS models [23], so as to allow them to more adequately deal with data and knowledge management requirements of advanced information management applications, especially WFMSs. The architecture of ADOME is characterized by the seamless integration of a rule base, an OODBMS and a procedure base. Role extension to the OO model has been employed for accommodating dynamic nature of object states and for capturing more application semantics at the knowledge level. Roles also act as "mediators" for bridging the gap between database, knowledge base and procedure base semantics, and facilitating dynamic binding of data object with rules and procedures [22]. Advanced ECA rules are also supported [5,6]. The following components / enabling technologies of ADOME are useful for various aspects of a WFMS, especially facilitating reuse:

- *Object-oriented database (OODB)* – Using OODB for the modeling and processing of complex objects and their relationships is almost a consensus for building an

advanced WFMS and other next generation information systems. For example, the composition hierarchy for modeling activities, sub-activities down to tasks and isa hierarchy for PSAs not only captures more semantics than traditional relational models, but also helps in reuse of their definitions in the WFMS. It also enables easier maintenance, understandability and extensibility than the large number of inter-related tables [19]. Moreover, the OO paradigm enables flexible passing of different forms of data among agents and tasks, with the OODB providing for a convenient general persistent storage for almost everything in the WFMS that requires to be recorded.

- *Roles* – Roles enables PSA objects to be dynamically associated with one or more different functions, responsibility and authority. It also captures attributes, states, methods and knowledge specific to individual positions/agents rather than the PSA player objects. With roles extended with multiple-inheritance, capabilities and roles for PSAs can be much better represented in a hierarchy (c.f. [7]).
- *Rules* – Declarative exception handlers in the form of Event-Condition-Action (ECA) rules enables automatic execution and exception handling based on events, where the *exceptions* correspond to the event part, while the *handlers* correspond to the condition-action parts. Rules can also be used for processing declarative knowledge such as organization policies, agent selection criteria, exception handling criteria, etc.
- *Flexibility of objects and schema* – This facilitates exception handling since real-time modification to objects, roles, rules and even workflow evolution are required during execution of workflow.

One of the main objectives of ADOME-WFMS is to provide extensive possibilities for reuse. In addition to the reuse of workflow definitions [7,8], we present in this section the modeling for exceptions, handlers and constraints that facilitate reuse. The ADOME prototype has been built by integrating an OODBMS (ITASCA [16]) and production inference engine (CLIPS [12]). Therefore, a WFMS can be implemented on top of it with relative ease. The architecture and functional aspects of the resultant ADOME-WFMS are as follows (cf. Fig. 1):

- *ADOME active expert OODBMS* (not shown in figure) provides a unified enabling technology for the WFMS, viz., object and role database, event specification and execution, rule / constraint specification and processing.
- *Activity Decomposer* facilitates the decomposition of activities into tasks. The user provides the knowledge and related data to decompose activities into tasks by a user interface.
- *Organizational Database* manages data objects for the organization, as well as PSA classes, instances and their capability token (role) specifications. Besides maintaining user-specified extensional tokens / roles systematically, intensional token/role derivation for a PSA is also supported.
- *Activity Executor* coordinates execution by user-raised and database generated events.

- *Match Maker* selects PSAs for executing tasks of an activity according to some selection criteria. ([8] describes the details in capability modeling and the mechanisms of the match maker of ADOME-WFMS.)
- *Exception Manager* handles various exceptions by re-executing failed tasks or their alternatives (either resolved by the WFMS or determined by the user) while maintaining forward progress.
- *Recovery Manager* performs various housekeeping functions, including rollback to maintain the WFMS in consistent states. (Details can be found in [9])

2.1 Exception Handling Mechanisms of ADOME-WFMS

In this paper, we shall concentrate on using a centralized control and coordination execution model centered on the *Activity Executor* of the WFMS. The *Activity Executor* monitor task execution status and enforces deadlines. For the normal task life cycle, it initiates the PSAs to be selected by the *Match Maker* to carry out their assigned task and get the response (if any) from the PSA upon task completion. On the other hand, if a task raises exception events or does not respond within the deadline (i.e., *time out*), the *Exception Manager* will respond and handle it.

2.1.1 Normal Workflow Execution

An event driven activity execution model with meta-ECA-rules can be found in [8]. Moreover, this provides a unified approach for normal activity execution and exception handling [8,19]. The mechanisms of ADOME-WFMS activity execution (cf. Fig. 1) are explained as follows:

- There is an *Activity Decomposer* module, which generates ECA rules for automatic coordination during the execution of workflow and stores them in the database [8,19].
- Users and external applications can trigger the corresponding start-events to start work.
- Upon a start-event, if the activity is a composite one, the activity executor will raise a start-event for the first sub-activity. This process will continue recursively downward the composition hierarchy until a leaf task is reached.[1] The activity executor invokes the *Match Maker* to select the appropriate PSA(s) for the task and then initiates the task. (Algorithm of the *Match Maker* is detailed in [8])
- The selected PSA will acknowledge or reject the assignment by raising a corresponding reply event.
- After finishing, the assigned task successfully, the PSA replies to the activity executor by raising a finish-event. The *Activity Executor* then carries on with the next step according to the result passed back.
- Upon failures or time out, the PSA or the system will raise an appropriate exception event to invoke the *Exception Manager*.

---

[1] This approach can handle dynamic resource allocation, online modification of workflow and exceptions in a rather flexible manner.

2.1.2 Detection of Events and Exceptions

The data dependency, temporal dependency and external input dependency, can be expressed by means of a uniform framework of events, such as *Data operations, Workflow, Clock Time, External Notification, Abstract Events.* Besides primitive events, any (recursive) combination of conjunction, disjunction, or sequence of other events can define a composite event. These events are all detected by the underlining ADOME event facilities as described in [5,6]. Since exceptions are ADOME events, detection of exceptions for ADOME-WFMS is well supported at run-time:

- External exceptions – events raised by external entities can be intercepted by the WFMS. These external events must be *apriori* characterized as events generated due to exceptions.
- Workflow exceptions raised by WFMS components, e.g.:
  Match Maker – cannot find PSA
  Activity Executor - PSA reject assignment, not enough resources
  Organization Database – data constraint violations upon update
  Exception Manager – (ignored) failure of task / sub-activity will cause exception to its parent
- Workflow exceptions detected by automatic ADOME ECA rules and/or constraints, e.g.:
  Activities cannot meet deadline
  Activities constraint violation (e.g. budget exceeded)

2.1.3 Handling Exceptions

As supported by the underlying ADOME facilities, the following information will be passed to the *Exception Manager* during the exception:

- source and type of exception,
- for workflow exceptions, state information of the task / activity [8],
- any extra parameters defined by the exception type (e.g., budget value).

The *Exception Manager* then takes control and carries out the following:

1. Perform notification if necessary.
2. Identify the appropriate exception handler(s) and execute them. Handles are modeled as sub-activities in ADOME-WFMS. One or more handlers will be executed until the problem is solved (c.f. Section 3).
3. If no appropriate exception handlers are found (i.e., an unexpected exception), or human intervention is specified, the *Human Intervention Manager* (c.f. Section 4) will be invoked. The human can then select the appropriate handler and/or perform workflow evolution (c.f. [10]).
4. If rollback is required, the *Recovery Manager* will be invoked for compensating activities (c.f. [9]).
5. Resume / redo execution, jump to the appropriate step as decided by step 2. or 3., or abort the current task / sub-activity so that the exception propagate to its parent for further handling. Though failure may propagate up the activity composition hierarchy, this approach localizes exception and thus reduces loss of work done.

3 Handling Expected Exceptions in ADOME-WFMS

In this section, we first discuss how exception handlers are identified and executed in ADOME-WFMS with the example in Fig 2. Then we discuss in depth how reuse is facilitated with advanced exception modeling based on the ADOME mechanisms.

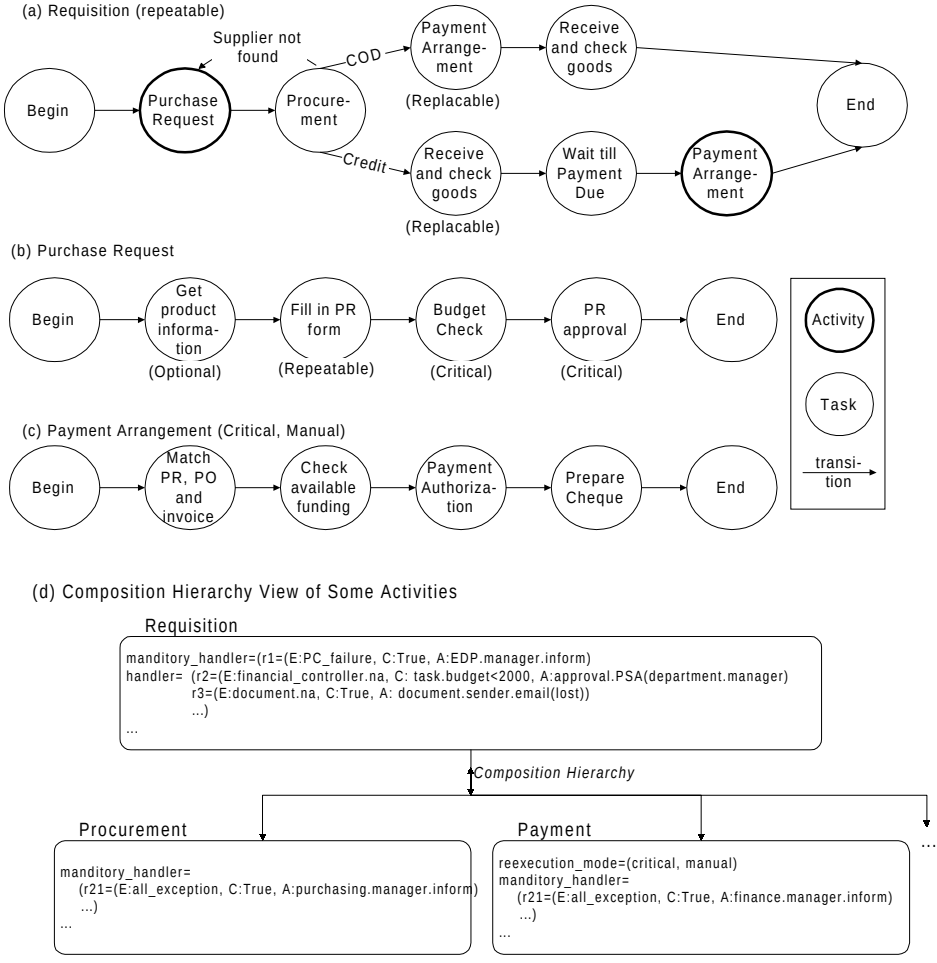

Fig. 2. Example Workflow of Requisition Procedures

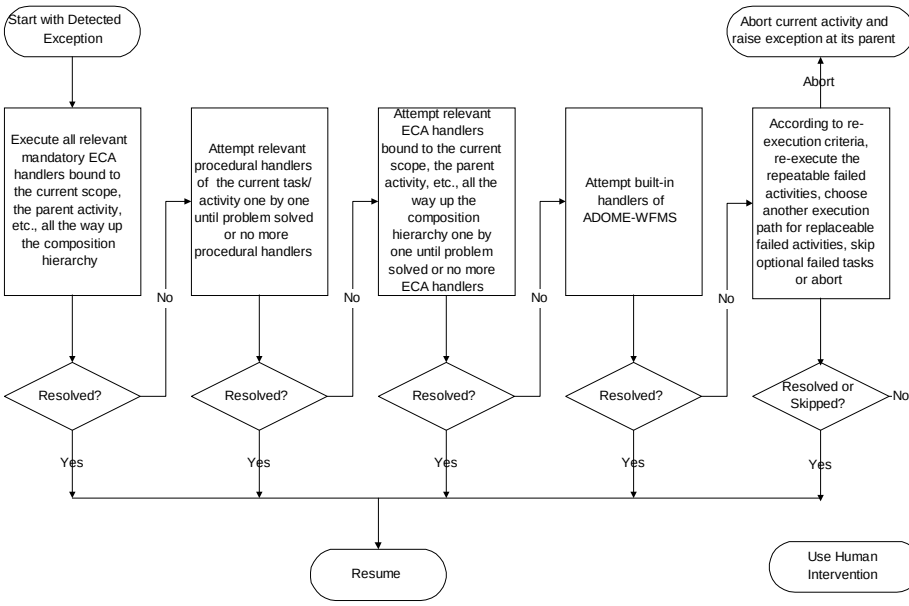

Fig. 3. Flowchart for identifying and Executing Exception Handlers

3.1 Identifying and Executing Exception Handlers

One or more exception handlers may be qualified to handle an exception that occurs. As illustrated in Fig. 3, the ADOME-WFMS Exception Manager employs the following priority order for selecting the appropriate exception handler:

1. Mandatory ECA handlers **-** Since the users specify these as mandatory, all relevant handlers (with event matched and condition fulfilled) bounded to the current scope are executed in the order of the task / sub-activity, its parent and all the way up to the *global* activity. For example, a mandatory ECA rule specifies that all exceptions should notify the purchasing manager in the procurement activity (*r21* in Fig. 2(d)) while a global mandatory ECA rule specifies all PC failures should be reported to the EDP manager (*r1* in Fig. 2(d)). In this case, the failure of a PC in the purchasing department causing an exception will trigger both rules and inform the purchasing manager and the EDP department.

These mandatory actions may or may not solve the problem causing the exception, such as, logging and notification. If they cannot solve the problem, other categories of exception handlers will be executed. For example, all exceptions in the 'Payment arrangement' activity (cf. Fig. 2(c)) are regarded as having severe consequences, so the financial manager will be informed (*r31* in Fig. 2(d)) and manual handling of exception is required.

Furthermore, the organization manager may later find that it is useful for all exception in a department be reported to its manager. So rules *r21* and *r31* are com-

bined and generalized as a new global mandatory rule *r4*=(E: all_exception, C: True, A: task.department.manager.inform).

2. Procedural handlers **-** These are extra branches for exception handling. Each procedural handler is specific to a certain task or sub-activity under a particular context for handling specific outcomes. Since they are explicit and context sensitive, they are chosen before (3) ECA handlers. For example, the 'supplier not found' arc (cf. Fig. 2(a)) represents a procedural handler.

3. ECA handlers **-** These are searched from the current activity up the composition hierarchy to allow special exception handlers to override default exception handlers if necessary. If more than one relevant handler was declared for the same activity, the one(s) for more specific exception type would be chosen over the more general exception type (as explained in Section 4). For example,

- The rule "If financial controller is not available, the department manager can approve any task involving less than $2000" (*r2* in Fig 2(d)) will enable the department manger to approve small purchase requests if the financial controller is not available.
- The rule "Send Email to the issuer if a document is lost" (*r3* in Fig 2(d)) will cause sending of an email to the supplier if an invoice is lost in the step 'Match PR, PO and invoice'.

4. Built-in handlers **-** For generic exceptions, ADOME-WFMS has built-in exception handlers, such as:

- 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.

ADOME-WFMS supports a lot of exception handling resolutions relating to PSA assignment based on capability matching, such as amending the capabilities of PSAs and changing capability requirements for a task instance [8]. This is important because significant portions 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"[8]. This increases the chance of finding suitable PSA(s) automatically (thus avoiding PSA not available exception), and finding alternate PSAs for repeating a task upon exception. For automatic switching of PSA assignment among tasks, the above-mentioned capability processing features are vital to the success of this resolution scheme. It should be noted that quite a number of traditional WFMSs like Flowmark [1] and OASIS [24], do not readily support or employ the notion of *capability matching* for PSA assignment to tasks.

5. Re-execution criteria **-** ADOME-WFMS can resolve and decide for the correct alternative PSAs or alternate execution branch automatically if the re-execution pattern for a task has been specified. This feature can save many tedious explicit jumps and aborts, especially with scoping in ADOME-WFMS (cf. Section 4). Moreover, this way one can resolve many unexpected exceptions if re-execution helps.

The WFMS will automatically re-execute the *repeatable* failed activities; choose another execution path for *replaceable* failed activities; skip *optional* failed tasks, if none of the above handlers is specified. However, *critical* failed tasks without explicit handlers are unexpected exceptions. (Section 3.2 discusses prevention of cascaded exceptions and loops.) Therefore, it will result in human intervention. Upon re-execution, in order to maintain work continuity and save starting up overhead, the same agent is preferred unless otherwise specified. The next candidate would be the nearest capable sibling or ancestor according to the organization structure (i.e., most probably a member of the team or the supervisor). For example:

- The 'Requisition' activity specified in Fig. 2(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. Fig. 2(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 costing around $500 and let the procurement department take care of the rest.) Hence, this task is optional (cf. Fig. 2(b)).
- As illustrated in Fig. 2(b), tasks 'Budget check' and 'PR approval' are flagged critical so that the sub-activity 'Purchase Request' will fail immediately if these tasks are not executed. However, 'Purchase Request' is repeatable so that the user can also revise the budget and /or product specification to retry for approval.

3.2 Handling Cascaded Exceptions and Loops

In order to handle cascaded exceptions effectively and avoid infinite generation of cascaded exceptions, the following *safety* measures are employed in ADOME-WFMS:

- Notification within the exception handling activity to the human deciding on the handler for an unexpected exception (e.g. to report even expected exceptions).
- If cascaded exceptions occur, the same human should be notified for better management and decisions.
- Tighter constraints, such as deadline and budget, can be introduced to avoid the exception handling activity to run indefinitely and let the control pass back for human decisions.
- When a human decides a jump back to a previously executed step as exception handler (including simple redoing of the currently failed task), there may be a potential danger of looping and the human will thus be warned.
- The *Activity Executor* will keep track of the sequence of executed tasks. If the same task in the same context is executed for over a certain number of times (or a specified iteration count), a warning or exception will be raised for human intervention or alternative actions.

- Backtracking so that the human can undo some of the decisions taken (if possible) and finalize on the resolution once all different aspects of exception handling are taken care of.

4 Exception Modeling for Reuse in ADOME-WFMS

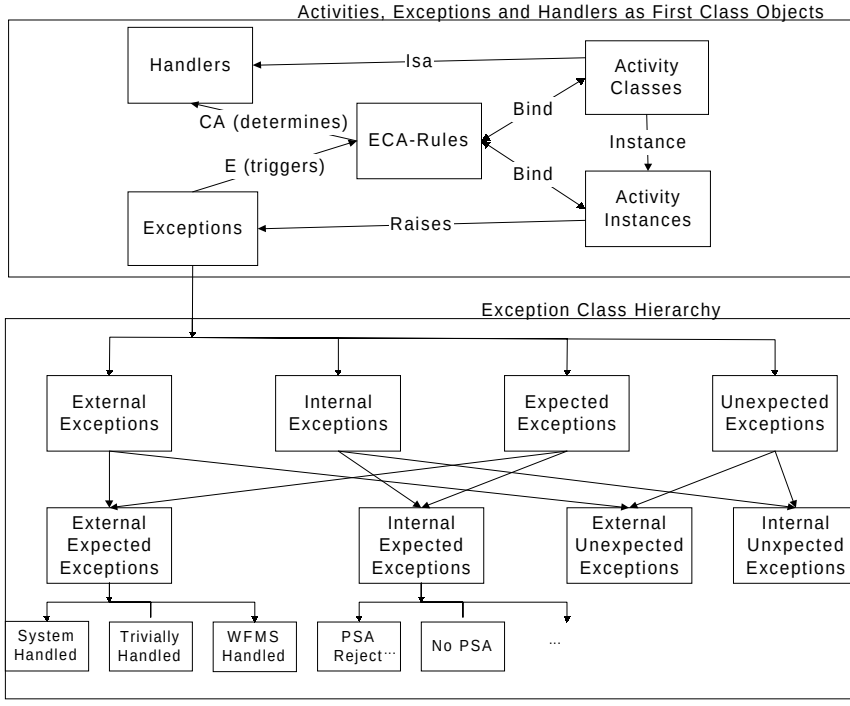

Figure 4. Activities, Exceptions, and Handlers as first-class objects

Figure 4 illustrates the main entities and relationships in ADOME-WFMS regarding to exception handling. These entities are all modeled as first-class objects. In particular, the class *exceptions* is a subclass of class *events.* Taxonomy of exceptions and handlers can be found in [8]. Handlers are modeled as sub-activities so that they can carry out any complicated actions; and nested exceptions are supported by recursive invoking of the *Exception Manager*.

In ADOME-WFMS, declarative exception handlers in the form of ECA rules can be bound to selected classes, objects and roles. Furthermore, handlers can be specified within the scope of different activity and sub-activity levels, i.e., the handler applies not only to the body of the target activity but also to all its sub-activities and tasks.

Similarly, human intervention requirements of exception handling (automatic, warning, cooperative 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.

The ADOME-WFMS *Human Intervention Manager* supports the user to modify all the above declarations and associations at run-time as described in [8,10]. The power of ADOME-WFMS in reuse over other systems (such as [3,4,11,19,20]) is mainly due ADOME's ability in dynamic binding of rules to different dimensions (objects, roles, sub-activities, etc.) at run-time as explained below. However, a methodology in workflow design to facilitate reuse of exception handlers is beyond the scope of this paper.

4.1 Reusing Exception Handlers

Since exceptions can be 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 at 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 each activity composition hierarchy (as explained in the previous section).
- Declarative exception handlers are first-class ECA rule objects. A rule object *r* is declared and defined once and then can be associated with more than one scope by repeated binding. (E.g., Bind *r9* to *payment*, *requisition*).
- 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 $\vee$ PC_failure, A: EDP.manager.inform), and increase the applicability of the exception handlers.
- Meta-level rules can be instantiated through parameters and supplied methods to specify rules, such as budget rules instantiated with actual budget figure.

4.2 Reusing Constraint Definitions

In order for modification of task instances and workflow evolution to be in accordance with users' requirement, we employ a strategy of associating consistency constraints with appropriate entities of different levels and dimensions as explained below. When constraints are violated at run-time, appropriate exceptions will be raised for handling. Since constraints are also implemented as ECA rules in ADOME-

WFMS, reuse and evolution of consistency constraints are both possible and similar to those of exceptions and exception handlers in general. Associating constraints to different levels and dimensions can be naturally specified with ADOME's facilities:

- Organization Composition Hierarchy - organization, units, sub-units and groups. Reuse is supported through inheritance down the composition hierarchy. The objective of the organization is also that of its units, sub-units and groups, while individual units can only have their more specific objectives not violating that of the organization.
- Activity Composition Hierarchy – projects (any collection of activities), activities, sub-activities, tasks. Reuse is supported through inheritance down the composition hierarchy. Deadlines and budgets are typical examples. Furthermore, scoping produces further flexibility for handling violation of constraints so that failed sub-activities (ignored exceptions) can be remedied by a higher-level activity.
- Role/Token Hierarchy – Positions and capability. Reuse is supported through multiple-inheritance down the hierarchy. For example, programmers (and hence senior programmers) are allowed to log on to the development computer account; however, the minimum productivity index of a senior programmer can be specified to be higher than that of a junior programmer by overriding rules as supported by ADOME [5,6].
- Any class and sub-classes of objects in ADOME-WFMS (such as PSA and resources).
- Combination of above levels and dimensions. As supported by ADOME [5,6], constraints and rules are first class objects and can be bound to objects, classes, and roles repeatedly. These can also be generalized or specialized by the 'or' and 'and' connectors. For example, programmers, engineers and managers should be university graduates.

4.3 Reusing Compensation Transactions for Workflow Recovery

Compensation transactions are considered as part of the exception handling procedure and are modeled as sub-activities of the exception handlers in ADOME-WFMS. In fact, the execution and definition of compensation transaction are implemented as ECA rules, which corresponds to the integrated framework of event-driven activity execution in ADOME-WFMS. This allows different compensation transactions to be executed according to the exception type and general conditions under execution [9]. The reuse of definitions of compensation transactions follows from its rich set of features in reuse of exception handlers:

- Once a compensation transaction is defined, they can be used in different contexts.
- "*Simple rollback*" (restoring the previous value or state of the object like handling traditional database update in workflow) and "*mark void*" (keeping useful documents or partial work done for reference) are implemented as pre-defined compensation transaction.
- The activity decomposition hierarchy also provides scoping for associating compensation transaction with a sub-activity tree.

- A compensation transaction can be bound to each rollback class and sub-class of objects (also for roles and sub-roles).
- Dynamic use of compensation transactions according to different exception events can be facilitated with the flexibility in event definition: event isa-hierarchy and event 'or' composition.

5 Related Work

A classical article on exception handling is [2], which focused on database aspects of exception and handling techniques instead of workflow systems. On the other hand, notable advanced WFMSs have been developed in the past years [1, 4, 13, 18, 19, 21, 24, 25]. Among them, TriGSflow [16] and the work of Kumar et al [18], perhaps have the closest basic design with ours in that it adopts an OO design, and utilizes rules and roles. However, they did not address a variety of exception conditions or use capability matching for tasks and agents.

Ellis, et al [14] is among the earlier work in workflow evolution. WIDE [4] used object and rule modeling techniques and suggested some measures in handling exceptions. Exceptions were classified but not handling approaches. They also addressed reuse and management of exception handlers with implementation details, but not adequately considered high level semantics, especially inter-relationship among entities in a WFMS. Their workflow evolution primitives were not at a semantic level. PROSYT [11] addressed inconsistencies and deviations in general process support systems (where WFMS is considered as a kind of process support systems), but the contribution was more on the formal modeling than semantic modeling. [3] presented a framework for tolerating exceptions in WFMS close to ours [8], but without details in logical and implementation levels. [20] adopted a knowledge-base approach to handling exceptions in WFMS, with strong emphasis on agent management. [26] studied the categories of exceptions in WFMS but did not involve workflow evolution.

Flowmark uses Sagas and flexible transactions for modeling workflow exception handling; and its extension, Exotica/FMDC [1], handles disconnected agents. Since Flowmark only finds out all possible candidates for task execution and then lets them volunteer for the execution instead of using capability matching, effectiveness and fairness may be impaired. WAMO [13] also uses Sagas and flexible transactions for supporting workflow exception handling. It also offers a preliminary classification of exceptions in which we have made some extensions in our taxonomy of exception categories (cf. [8]). Similarly, other works like [15], ConTract [25] and OPERA [17] focus on transactional aspects and thus on lower level issues.

In summary, other workflow systems either do not address exception-handling problems comprehensively or concentrate only on extended transaction models. Furthermore, few systems have advocated (let alone supported) an extensive meta-modeling approach (based on PSAs, match-making, exception handling, etc.). Compared with the systems close to us, ADOME-WFMS has the most features to facilitate reuse.

6 Conclusion

This paper has presented adaptive exception handling in ADOME-WFMS, a flexible WFMS based on ADOME; an active OODBMS extended with role and rule facilities. Compared with other research on this topic, ADOME provides an improved environment for developing a WFMS, which can adapt to changing requirements, with extensive support for reuse. In particular, the resultant system (i.e., ADOME-WFMS) supports a rich taxonomy of exception types and their handling approaches, and a novel augmented solution for exception handling based on workflow evolution. Effective reuse of workflow definitions, exceptions, handlers and constraints in ADOME-WFMS has also been presented. This paper has also described in detail, how expected exceptions are actually resolved with the ADOME-WFMS Exception Manager. It should be noted that, though exception handling is highly automated in ADOME-WFMS by scoping, binding and reuse, human intervention management must be provided to support for (totally) unexpected exceptions and drastic workflow evolutions. ADOME-WFMS is currently being built on top of the ADOME prototype system, with a web-based user interface [10] to accommodate the whole range of activities.

References

1. G. Alonso, et al. Exotica/FMDC: a workflow management system for mobile and disconnected clients. *Distributed & Parallel Databases*, **4**(3):229-247 (1996).
2. A. Boridga, Language Features for Flexible Handling of Exceptions, *ACM Trans. on Database Systems* (1985).
3. A. Borgida and T. Murata, A Unified Framework for Tolerating Exceptions in Workflow/Process Models - A Persistent Object Approach, *International Joint Conference on Work Activities Coordination and Collaboration (WACC '99)*, San Francisco (1999).
4. F. Casati, G. Pozzi. Modeling and Managing Exceptional Behaviors in Workflow management Systems, Proceedings of CoopIS'99, Edinburgh, Scotland, September 1999
5. L. C. Chan and Q. Li. Devising a Flexible Event Model on top of a Common Data / Knowledge Storage Manager. In *Proceedings of 6th Intl. Workshop on Information Technologies and Systems (WITS '96)*, Cleveland, Ohio, pp.182-191 (1996).
6. L. C. Chan and Q. Li. An Extensible Approach to Reactive Processing in an Advanced Object Modeling Environment. In *Proceedings of 8th Intl. Conf. on Database and Expert Systems Applications (DEXA '97)*, LNCS(1308), pp.38-47, Toulouse, France (1997).
7. D. K. W. Chiu, K. Karlapalem and Q. Li. Developing a Workflow Management System in an Integrated Object-Oriented Modeling Environment. In *Proceedings of 6th Int'l Conf. on Sofware Engineering and Knowledge Engineering (SEKE'98)*, pp.71-78, San Francisco (1998).
8. D.K.W. Chiu, Q. Li and K. Karlapalem, "A Meta Modeling Approach for Workflow Management Systems Supporting Exception Handling", Special Issue on Method Engineering and Metamodeling, *Information Systems*, Elsevier Science, 24(2):159-184 (1999).
9. D.K.W. Chiu, Q. Li and K. Karlapalem, Facilitating Exception Handling with Recovery Techniques in ADOME Workflow Management System, *Journal of Applied Systems*

Studies, Cambridge International Science Publishing, Cambridge, England (2000 to appear).
10. D.K.W. Chiu, Q. Li and K. Karlapalem, A Web-based Interface for ADOME Workflow Management System Facilitating Exception Handling, submitted to WISE'2000.
11. G. Cugola, Inconsistencies and Deviations in Process Support Systems, *PhD Thesis, Politecnico di Milano* (1998)
12. http://www.ghg.net/clips/CLIPS.html
13. J. Eder and W. Liebhart. The Workflow Activity Model WAMO. In *Proceeding of CoopIS-95*, pp 97-98. (1995).
14. S. Ellis et al , Dynamic Change within Workflow Systems, *Proceedings of the Conference on Organizational Computing Systems* (1995).
15. D. Georgakopoulos, M. F. Hornich and F. Manola. Customizing Transaction Models and Mechanisms in a Programmable Envioronment Supporting Reliable Workflow Automation. *IEEE Transactions on Knowledge and Data Engineering*. **8**(4):630-649 (1996).
16. Ibex Corporation. *http://www.ibex.ch/*
17. C. Hagen and G. Alonso, Flexible Exception Handling in the OPERA Process Support System, *18th International Conference on Distributed Computing Systems* (ICDCS 98), Amsterdam, The Netherlands (1998).
18. G. Kappel, et.al. Workflow Management Based on Objects, Rules, and Roles. *IEEE Bulletin of the Technical Committee on Data Engineering* **18**(1)11-18 (1995).
19. K. Karlapalem, H. P. Yeung and P. C. K. Hung. CapBaseED-AMS - A Framework for Capability-Based and Event-Driven Activity Management System. In *Proceeding of COOPIS '95*, pp. 205-219 (1995).
20. Mark Klein and Chrysanthos Dellarocas, A Knowledge-Based Approach to Handling Exceptions in Workflow Systems, Proceedings of the Third International Conference on Autonomous Agents, Seattle, Washington (1999).
21. A. Kumar, et.al. A framework for dynamic routing and operational integrity controls in a workflow management system. In *Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences* **3**:492-501 (1996).
22. Q. Li and F. H. Lochovsky. Roles: Extending Object Behaviour to Support Knowledge Semantics. In *Proceeding of Int'l. Symposium on Advanced Database Technologies and Their Integration*, Nara, Japan, pp. 314-322 (1994).
23. Q. Li and F. H. Lochovsky. ADOME: an Advanced Object Modeling Environment. *IEEE Transactions on Knowledge and Data Engineering*, **10**(2):255-276 (1998).
24. C. Martens and C.C. Woo. OASIS: An Integrative Toolkit for Developing Autonomous Applications in Decentralized Environments. *Journal of Organizational Computing*, New Jersey: Ablex Publishing Corporation, **7**(2&3):227-251 (1997).
25. A. Reuter and F. Schwenkreis. ConTacts - A Low-Level Mechanism for Building General-Purpose Workflow Management Systems. *IEEE Bulletin of the Technical Committee on Data Engineering* **18**(1)4-10 (1995).
26. Saastamoinen, H. T., On the Handling of Exceptions in Information Systems, Ph.D. Thesis, University of Jyväskylä (1995).