

WEB INTERFACE-DRIVEN COOPERATIVE EXCEPTION HANDLING IN ADOME WORKFLOW MANAGEMENT SYSTEM

DICKSON K.W. CHIU¹, QING LI², and KAMALAKAR KARLAPALEM³

¹Department of Computer Science, University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong

²Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong

³Indian Institute of Information Technology, Gachibowli, Hyderabad, India

(Received 30 September 2000; in final revised form 14 February 2000)

Abstract — 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, cooperative support for user-driven computer supported resolution of unexpected exceptions and workflow evolution at run-time is vital for an adaptive WFMS. We have been developing ADOME-WFMS as a comprehensive framework in which the problem of workflow exception handling can be adequately addressed. In this article, we present an adaptive exception manager and its web-based interface for ADOME-WFMS with procedures for supporting the following: reuse of exception handlers, thorough and automated resolution of expected exceptions, effective management of Problem Solving Agents, cooperative exception handling, user-driven computer supported resolution of unexpected exceptions, and workflow evolution. © 2001 Elsevier Science Ltd. All rights reserved

Key words: Workflow Management, Exception Handling, Reuse, Workflow Evolution, ECA Rules, Meta-Modeling, Object-Orientation, Workflow Recovery, Web Interface

1. INTRODUCTION

Workflow management system technology, though recent, has been regarded as one of the main types of advanced information systems. It is perceived that workflow technology not only requires the support for complex data model functionality, but also flexibility for dynamically modifying the workflow specifications, especially in cases of exception handling. Because of unanticipated possibilities, special cases, and/or changes in requirement and operation environment, exceptions may occur frequently during the execution of a business process. An exception is an event (i.e., something that happens) that deviates from normal behavior or may prevent forward progress of a workflow. Upon unexpected exceptions, a comprehensive WFMS should be able to support the users to reallocate resources (data / object update) or to amend the workflow, such as adding alternatives (workflow evolution). Further, frequent occurrences of similar exceptions have to be incorporated into workflow specifications as expected exceptions. Such workflow evolution can help avoid unnecessary exceptions by eliminating error-prone activities, adding alternatives, or by enhancing the operation environment. This can lead to a WFMS that supports workflow adaptation through exceptions.

In contrast with traditional software systems, workflows usually evolve more frequently, making reuse a vital issue. Reuse of workflow definitions and exception handlers are very important for the smooth operation of a flexible WFMS. Support for workflow evolution at run-time is vital for an adaptive WFMS. There have been a few WFMSs designed to address these two problems (viz. reuse issues and workflow evolution) effectively and adequately. However, none of them is based on a meta-modeling exception-centric approach.

With a meta-modeling approach, we can have a simple but expressive core data dictionary (meta-schema). From this meta-level schema, users can define all other classes for the WFMS, including activity schemas, exceptions and handlers. Further, from these schemas WFMS objects (in particular, activity instances) can be instantiated. This contributes a substantial improvement to WFMS modeling based on relational models because the entity modeling and implementation is tied together in a straightforward manner. Extensive reuse can also be facilitated as discussed in Section 5.

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 Execution

Manager to start an activity. The WFMS Execution Manager uses events to trigger execution of tasks, while finished tasks will inform the Execution Manager with events. A task is executed by a problem solving agent (PSA) which is a hardware/software system or a human being. Upon an exception, exception events will trigger the WFMS Exception Manager to take control of resolutions. These resolutions can trigger a software solution or will involve a human to cooperatively resolve an exception.

An effective user interface is vital to the execution of the above features. We choose to use a web-based user interface because workflow and agents tends to be widely distributed, even involving other organizations across the Internet. Mobile clients can be supported (alerted by ICQ or electronic mail) by the Internet, and they can access the WFMS with a web-browser. Direct message passing between clients and remote data sharing can be facilitated. On the other hand, web-based tools are a prevailing technology that has a wide range of utilities and off-the-shelf applications, supporting wide ranges of hardware and software platforms at a relatively low-cost.

In this regard, we have developed ADOME-WFMS, based-on an Advanced Object Modeling Environment (ADOME [45]), with a novel exception centric approach. The key features are as follows:

1. Effective coordination of PSAs and an object-oriented capability-based approach to match tasks and agents;
2. Automatic resolution of expected exceptions and exception driven workflow recovery;
3. Dynamic binding of exception handlers to activities with scoping, and to classes, objects and roles;
4. Addition, deletion and modification of exception handlers at run-time through workflow evolution;
5. Specifying and reusing exception handlers upon unexpected exceptions and system-assisted exception handling; and
6. Application of workflow evolution and workflow recovery in exception handling.

Thus, adding a web-based user interface allows ADOME-WFMS to effectively support distribution of PSA and workflow execution even with a centralized control design. In this article, we present a framework for flexible workflow enactment and online workflow evolution in an advanced object environment (active OODBMS with role and dynamic schema support), with reference to ADOME-WFMS. More details regarding classification of exceptions and handlers, and modeling aspects for ADOME-WFMS are given in [22]. In addition, ADOME-WFMS exception driven workflow recovery has been presented in [26]. The objectives and contribution of this article include: (i) the mechanism of ADOME-WFMS and resolution of expected exceptions, (ii) support of reuse for workflow definitions, constraints, exception types and handlers in ADOME-WFMS, (iii) an augmented solution for exception handling based on workflow evolution, (iv) management of distributed PSA and allow for distributed users, (v) user-driven computer supported resolution of unexpected exceptions, and (vi) demonstrating the use of ADOME-WFMS in supporting exception handling through effective web interface facilities.

The rest of our article is organized as follows. Section 2 presents a meta-modeling approach to activity modeling, which facilitates reuse and workflow evolution. Section 3 presents the architecture of ADOME-WFMS with web-based PSA coordination and general mechanisms. Section 4 discusses how ADOME-WFMS resolves for expected exceptions. Section 5 explains how reuse can be facilitated. The ADOME-WFMS Human Intervention Manager is detailed in Section 6 to illustrate how unexpected exceptions are handled in ADOME-WFMS, with novel web-based workflow evolution functions. Section 7 compares related work. Finally, we conclude the article with our plans for further research in Section 8.

2. FLEXIBLE ACTIVITY META-MODELING

In this article, 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 Figure 1). A task is usually handled by a *single* PSA. The WFMS 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 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,

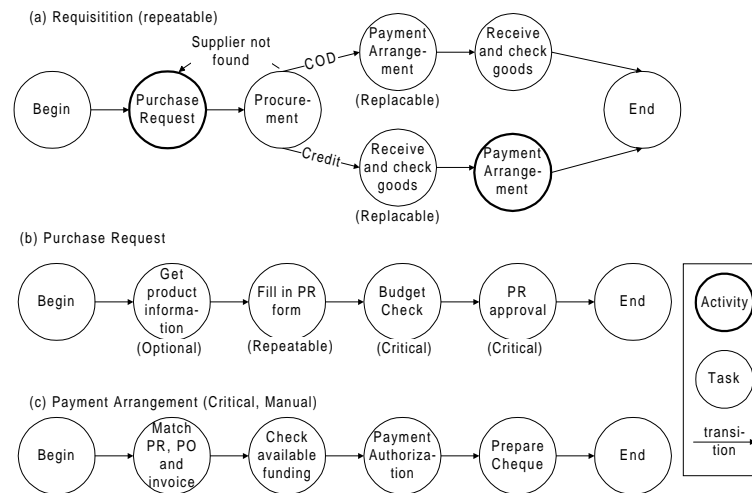


Fig. 1: Example Workflow of Requisition Procedures

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.

The activity model of ADOME-WFMS is basically in accordance with the WfMC standard [57, 58]. Typically, the schema of an activity (workflow) is recursively decomposed into sub-activities and eventually down to the unit level called tasks. Sibling sub-activities / tasks belonging to the same parent activity form a directed graph that defines the execution dependencies among them. These dependencies, including sequence, parallel, conditional and synchronization, are expressed graphically as follows. An arc pointing from activity A1 to A2 denotes A2 is to be executed immediately after A1 (i.e. A2 is a successor of A1). Outgoing arcs from activity A to more than one successor denote parallel execution branches of all the successors after A is completed (called split). Transition predicates may be associated with these splits. Only those arcs where transition predicate evaluates to true are executed. If the transition predicates of a split are in mutual exclusion, the split is called an OR-split (representing decision), otherwise it is called an AND-split (representing parallel execution). Incoming arcs towards an activity A from more than one predecessor are called join. AND-join synchronization activates A when all predecessors of A finish. OR-join activates A when any predecessor finishes (i.e., no synchronization is involved).

Figure 1 illustrates a requisition workflow, which is used as an example in the rest of the article. The requisition workflow is composed of the sub-activities “purchase request”, “procurement”, “payment arrangement” and task “receive and check goods”. If the purchase order is on cash-on-delivery (COD) terms, the order of execution is “payment arrangement” and then “received and check goods”. Otherwise, if the purchase order is on credit terms, the order is “receive and check goods” first and then “payment arrangement” upon payment due. This decision is represented with an OR-split and OR-join. The “purchase request” and “payment arrangement” sub-activities are further composed of other tasks, like “get product information”, “fill in PR form”, etc., as illustrated.

Many of the earlier WFMSs [29] were built on top of traditional database technologies (e.g., relational databases). They fall short in facilitating / offering flexibility of modeling, ease of implementation, and/or in handling dynamic run-time requirements. Advanced features — objects, rules, roles, active capability and the flexibility — of object-oriented database systems are needed to facilitate the development of a versatile WFMS, especially with meta-modeling approach. In ADOME-WFMS, we advocate a three-level meta-modeling approach wherein workflows, capabilities, exceptions, and handlers are defined at a meta-level as depicted in Figure 2. Workflow templates are defined at the meta-level so that actual workflows can be instantiated for specific applications. For example, a generic requisition workflow

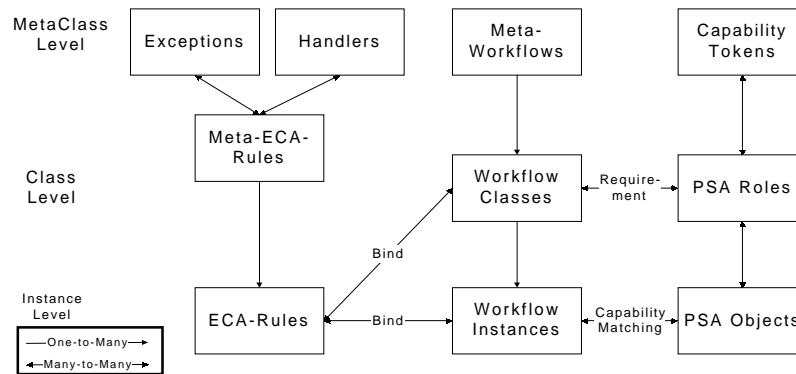


Fig. 2: Three-Level Meta-Modeling for ADOME-WFMS

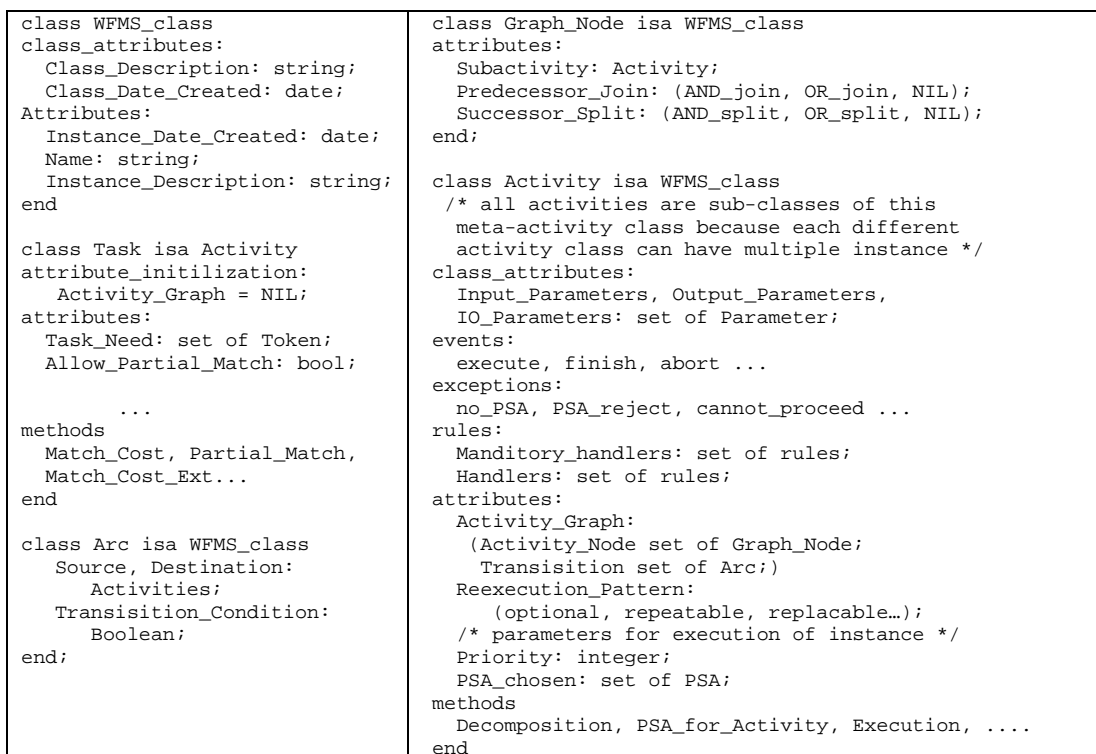


Fig. 3: Meta-Level Specification of Activities and Tasks

template can be declared at the meta-level, so that specific requisition workflows have customized rules and sub-activities can be instantiated (cf. Figure 2). Capability tokens are defined at the meta-level so that they can be combined to form PSA-roles, which capture requirements of task classes (cf. [22]). Exceptions (which are events) and handlers (which correspond to conditions and actions) are defined at the meta-level. Exceptions are associated to handlers in the form of meta-Event-Condition-Action-rules (meta-ECA-rules). Specific ECA-rules can then be bound to workflow for versatile exception handling (cf. Section 5).

The meta-level design for activity schemas is shown in Figure 3 with the following features. All activity schemas are treated as sub-classes of the meta-class *Activity*. Class-attributes (which is a feature supported in many advanced object-oriented systems [37]) are used for storing either attributes of the class object (such as class description) or attributes of the same value among all objects of the class (such as the input / output specifications). All sub-class objects inherit the definition attributes of the super-class object but each of the classes can have their own value of class attributes. WFMS related

events for activities (such as `execute`, `finish`, `abort`), standard workflow exceptions (such as `no_PSA`, `PSA_reject`, `cannot_proceed`) are declared with the meta-class so that these features are applicable to all activities schemas. The meta-activity schema contains all features for defining activity schemas, such as input/output parameters, and the activity graph, which describes the sub-activities and their incoming/ outgoing transitions, join/split types, mandatory/regular handlers. `WFMS_class` serves as the root class of all class definitions in the WFMS for easy maintenance of the classes and objects.

Each activity needs to be decomposed once by the user, after which this decomposition is stored and available for reuse. Thus, a sub-activity may be part of many other activities. Users can easily compose more complicated new activities based on existing ones. Update and refinement to the decomposition is thus efficient and effective. In ADOME-WFMS, we allow tasks to be decomposed later into sub-activities / tasks, rendering itself a complex activity. Thus, a top-down stepwise-refinement approach is also supported in parallel with a bottom-up composition approach.

The hierarchical composition is important for encapsulating details of activities and sub-activities so as to facilitate: (a) reuse of task and sub-activity definition in other new activities; (b) stepwise refinement by decomposing an activity into sub-activities representing more elementary tasks, if this is required; (c) easy maintenance for activity definition and exception handler definitions; (d) scoping for exception handlers; (e) the use of nested transaction models for execution control; (f) localizing failures and thus limiting the loss of work done; (g) capturing exceptions between task executions.

A hierarchical view of the requisition activity (from Figure 1) is presented in Figure 4, illustrating input/output specifications and rules bound to various activities. Figure 5 illustrates the sample declarations generated (by the decomposition method) from the requisition activity, according to the meta-activity schema. When an activity instance is started, it has its own copy of the activity graph and re-execution pattern. This allows both instance and schema level workflow evolution. On the other hand, rule objects can be declared outside of the scope of activities first, and then bound to individual activity schemas (or specific instances) to facilitate reuse.

This full object-oriented approach enables full inheritance of various activity properties (such as rules and re-execution mode) down the composition hierarchy and applies to each of the activity / task instances. From this meta-level schema, users can define all other classes for the WFMS, including activity schemas, PSAs, roles, exceptions and handlers. Further, from these schemas WFMS objects (in particular, activity instances) can be instantiated. This contributes a substantial improvement to WFMS modeling based on relational models (such as [41]) because the entity modeling and implementation is tied together in a straightforward manner. Extensive reuse is also facilitated as discussed in [19].

3. ADOME-WFMS ARCHITECTURE AND ACTIVITY ENACTMENT MECHANISM

In this section, we present the architecture of ADOME-WFMS and illustrate how activity enactment can be facilitated. 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 [45]. Advanced ECA rules are also supported [15, 17, 18]. The ADOME prototype has been built by integrating an OODBMS (ITASCA [37]) and production inference engine (CLIPS[27]). Therefore, a WFMS can be implemented on top of it with relative ease. The following components / enabling technologies of ADOME are useful for various aspects of a WFMS:

- *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 agents 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 table’s [41]. 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.

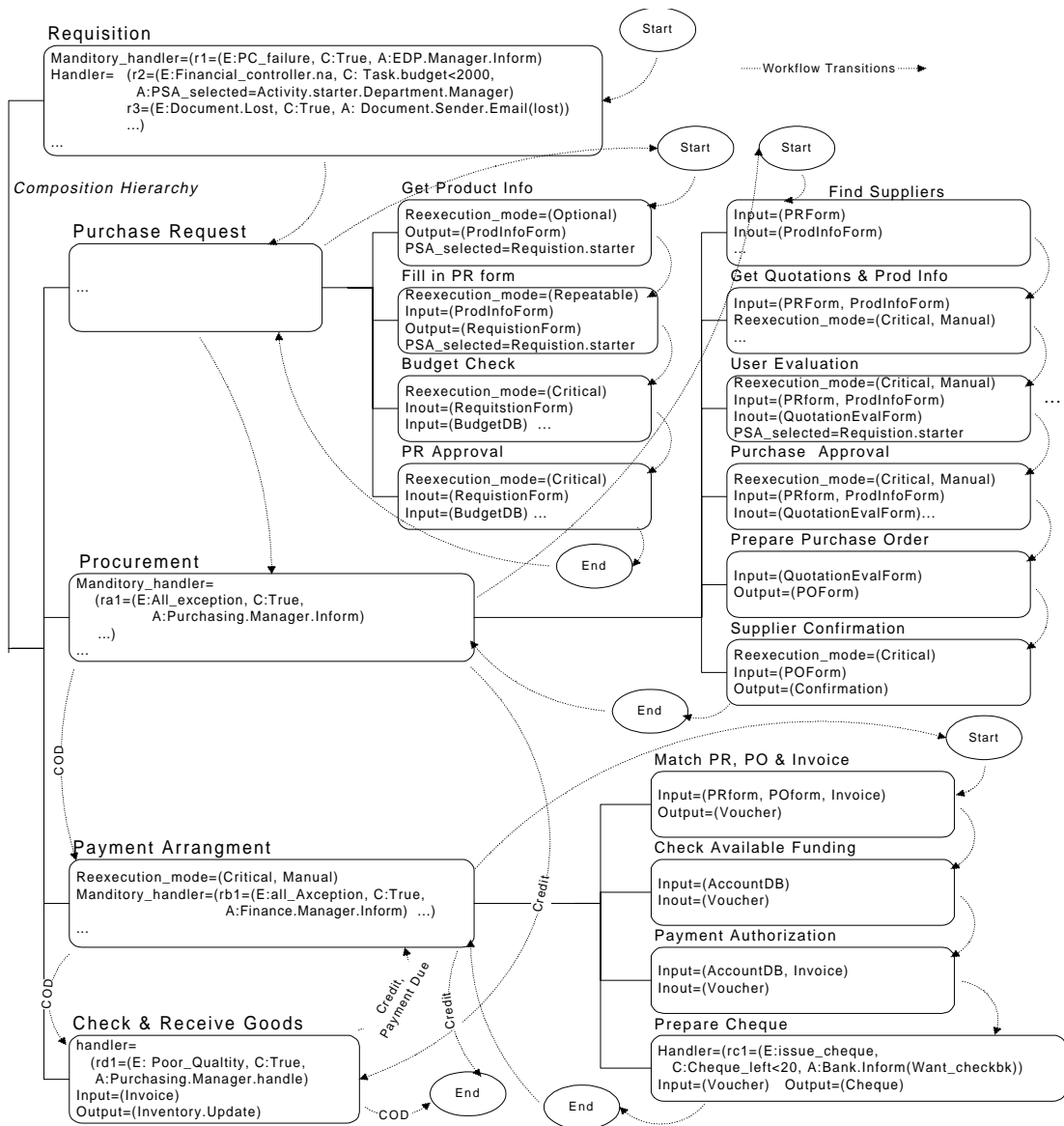


Fig. 4: Composition Hierarchy View of Requisition Procedures

- **Roles** – Roles enable agent 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 agent player objects. With roles extended with multiple-inheritance, capabilities and roles for agents can be much better represented in a hierarchy (cf. [22]).
- **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.

```

r1=(E:PC_failure, C: True, A:EDP.manager.inform);
r2=(E:Financial_controller.NA, C: task.budget<2000,
    A:PSA.Department.Manager.Approval);
r3=(E:Document.NA, C: True, A:document.sender.email(lost));
ral=(E:All_exceptions, C: True, A:Purchasing.Manager.Inform);

/* to handle second occurrence of a sub-activity */
class Payment_Arrangement2 isa Payment_Arrangement;
class Check_and_Receive_Goods2 isa Check_and_Receive_Goods;

class Requisition isa Activity
event:
    Issue_cheque, ...
exceptions:
    PC_failure, Out_of_cash ...
rules: /* rule binding */
    Mandatory_handlers=(r1);
    Handlers=(r2, r3);
attributes_initalization:
    Activity_Graph=(
/* Graph Nodes */
        ((BEGIN, NIL, NIL), (Purchase_Request, NIL, NIL),
         (Procurement, NIL, OR_split),
/* first branch */
         (Payment_Arrangement, NIL, NIL), (Check_and_Receive_Goods, NIL, NIL),
/* second branch */
         (Check_and_Receive_Goods2, NIL, NIL), (Payment_Arrangement2, NIL, NIL),
         (END, Or_join, NIL))
/* Transition arcs */
        ((BEGIN, Purchase_Request, true),
         (Purchase_Request, Procurement, true),
         (Procurement, Payment_arangement, COD),
         (Payment_Arrangement, Check_and_Receive_Goods, true),
         (Check_and_Receive_Goods, END, true),
         (Procurement, Check_and_Receive_Goods2, Credit),
         (Check_and_Receive_Goods2, Payment_Arrangement2, Payment_Due),
         (Payment_Arrangement, END, true)
        ))
end.

class Procurement isa Activity
class_attributes:
    Input_Parameters=(PRForm); IO_Parameter=(ProdInfoForm);
    Output_Parameters=(POForm, Confirmation);
exceptions:
    No_supplier ...
rules: /* rule binding */
    Mandatory_handlers=(ral);
attributes
    Activit_Graph = ((BEGIN,NIL,NIL), (Find_Suppliers, ...)... (END,NIL,NIL))
        ((BEGIN, Find_Suppliers, TRUE),...);
    ...
end.

task Prepare_Cheque isa Task
class_attributes:
    Input_Parameters=(Voucher); IO_Parameter=nil;
    Output_Parameters=(Cheque);
rules: /* local ECA rule */
    rcl=(E:Issue_cheque, C:Cheque_left<20, A:Bank.Inform(want_checkbk));
attributes_initalization:
    Task_Need=(AC_clerk); Allow_Partial_Match=false;
end.

```

Fig. 5: Sample Schema Declarations from the Requisition Activity Hierarchy

3.1. Architecture

The ADOME system was developed to enhance the knowledge-level modeling capabilities of OODBMS models [45], so as to allow them to more adequately deal with data and knowledge management requirements of advanced information management applications, especially WFMSs. The

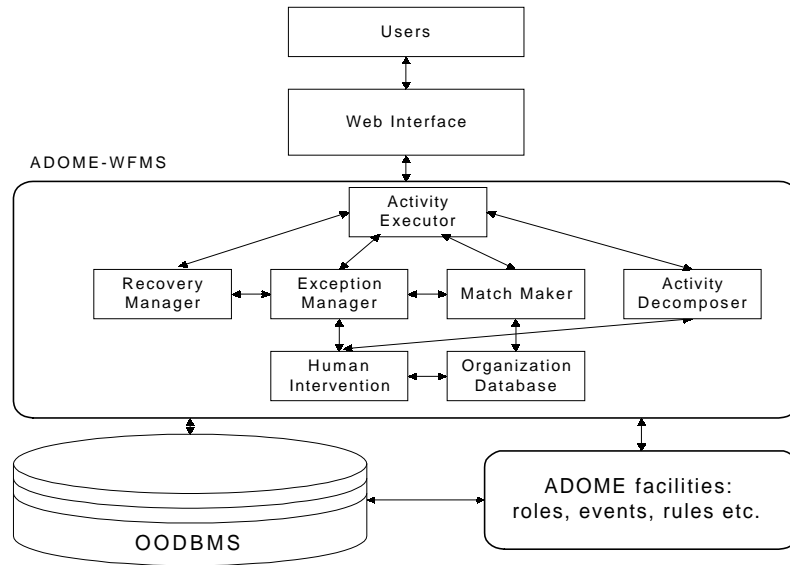


Fig. 6: ADOME-WFMS Architecture

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 the 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 objects with rules and procedures [45]. Advanced ECA rules are also supported [16, 18]. One of the main objectives of ADOME-WFMS is to provide extensive possibilities for reuse (cf. Section 5) and system-assisted exception handling (cf. Section 6).

The ADOME prototype has been built by integrating an OODBMS (ITASCA [37]) and production inference engine (CLIPS [27]). Therefore, a WFMS can be implemented on top of it with relative ease. In the case of ADOME-WFMS, the architecture and functional aspects are as follows:

ADOME active expert OODBMS provides a unified enabling technology for the WFMS, viz., object and role database, event specification and execution, rule / constraint specification and processing [45], etc. *Organization 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 (cf. [19]).

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 (cf. [22]). *Activity Executor* coordinates execution by user-raised and database generated events as explained in the next section. *Match Maker* selects PSAs for executing tasks of an activity according to some selection criteria as explained in [19]. *Web Interface* allows users and PSAs to access ADOME-WFMS and the database through a web-browser, so that the users can be mobile. This supports effective management of problem solving agents, system-assisted exception handling, user-driven computer supported resolution of unexpected exceptions, and workflow evolution.

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 as explained later in this chapter. *Recovery Manager* performs various housekeeping functions, including logging and rollback, to maintain the WFMS in consistent states as explained in [26].

3.2. Workflow Enactment Mechanisms of ADOME-WFMS

In this article, we shall concentrate on using a centralized control and coordination execution model centered on the *Activity Executor* of the WFMS. Figure 7 summarizes the control flow of task / activity execution, together with exception handling.

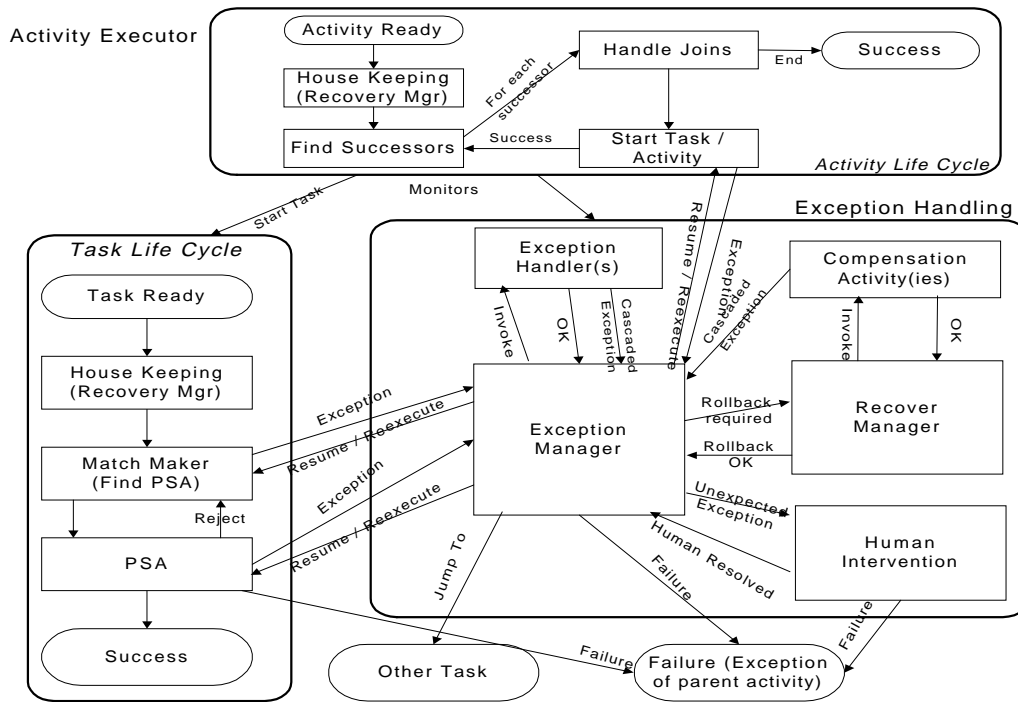


Fig. 7: ADOME-WFMS Task Execution, Exception and Recovery Control Flow

The *Activity Executor* monitors task execution status. 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 *time out* period, the *Exception Manager* will respond and handle it.

An event driven activity execution model with meta-ECA-rules has been described in the previous chapter. Moreover, this model provides a unified approach for normal activity execution and exception handling. Now, ADOME-WFMS can support activity execution through the *Web Interface*. Users can define an activity class with the ADOME-WFMS *Activity Editor*.

With a web-browser, a user can log on to ADOME-WFMS to search / browse for an activity class (by name, department, etc.). The system provides a preliminary checking function, which determines if the required PSA and resources are possibly available. (Their actual availability depends on the execution of other concurrent activities.) Pressing the *Start* button on the web page will trigger the corresponding start-event for the activity through the *Web Interface*. External application software can also trigger the corresponding start-event to start the workflow.

For human PSAs, the selected PSA will be informed via ICQ with the URL of an *Interface Page* generated from a template in the database by the *Web Interface*. Figure 8 shows the interface page for the 'purchase approval' task of the example requisition workflow (cf. Figure 1). If the PSA is offline or does not reply within a deadline, an electronic mail will be sent instead. The selected PSA can then use a web browser to open the interface page (logging on is necessary) to: (i) accept the assignment by pressing the *Accept* button and then start work later with the *Start Work* button, (ii) accept and start work immediately with the *Start Work* button (iii) reject the assignment with the *Reject* button. ADOME-WFMS will assume the PSA rejects if no reply can be received within a user-defined deadline.

The *Interface Page* contains also a list of required data objects for the task (such as other forms, voice or typed messages of colleagues, relevant database information, etc.). The PSA can then view or update the objects during his work. It should be noted that the PSA could go over the *objects passed* before deciding to accept the task.

After finishing the assigned task successfully, the PSA replies to the *Activity Executor* by pressing the *Normal Finish* button of the interface page. Data objects are passed via the mechanism of web-page forms (e.g., by filling in a web-based requisition form). The *Activity Executor* then carries on with the next step

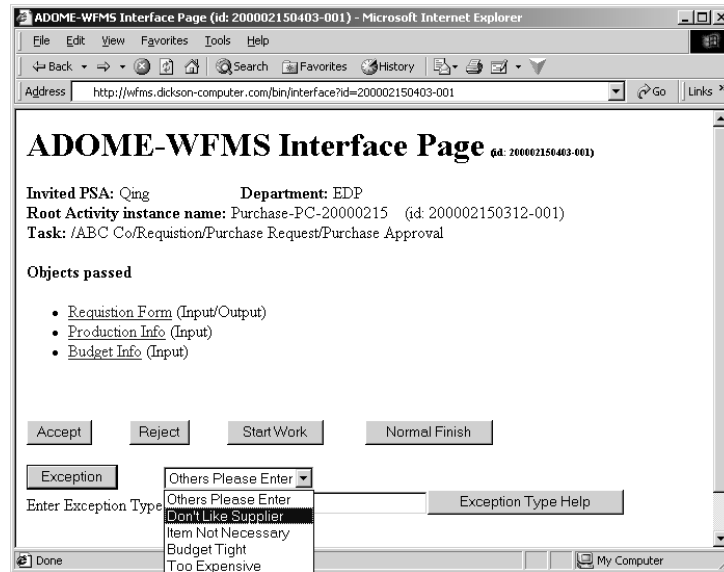


Fig. 8: ADOME-WFMS Interface Page

according to the result passed back. The PSA can report failure by choosing an exception type with the help of another pop-up page. Upon failure or time out, the *Exception Manager* will be invoked to handle the exception.

3.3. 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 underlying ADOME event facilities as described in [15, 17, 18]. Since exceptions are ADOME events, detection of exceptions for ADOME-WFMS is well supported at run-time.

External exceptions raised by external entities can be intercepted by the WFMS. These external events must be *a priori* characterized as events generated due to exceptions.

Changes in workflow definitions, rules, schema and any WFMS data objects are detected as update events, supported by the underlying ADOME active OODBMS mechanisms.

Workflow exceptions are raised by WFMS components, for example, PSA not available in *Match Maker*, not enough resources or PSA reject assignment in the *Activity Executor*, data constraint violations upon updating the *Organization Database*, (ignored) failure of task / sub-activity causing exception to its parent in the *Exception Manager*. Moreover, workflow exceptions are detected by automatic ADOME ECA rules and/or constraints, e.g., activities cannot meet deadline, activities constraint violation (e.g. budget exceeded).

3.4. Handling Exceptions

As supported by the underlying ADOME facilities, the following information items are passed to the *Exception Manager* upon the exception detection: source and type of exception, state information of the task / activity [18], and any extra parameters defined by the exception type (e.g., budget value). The *Exception Manager* (cf. Section 4) then takes control and carries out the following:

1. Perform notification if necessary.
2. Identify the appropriate exception handler(s) and execute them. Handlers are modeled as sub-activities in ADOME-WFMS. One or more handlers will be executed until the problem is solved.

3. If no appropriate exception handlers are found (i.e., an unexpected exception), or human intervention is specified, the *Human Intervention Manager* will be invoked. The human can then select the appropriate handler and/or perform workflow evolution.
4. If rollback is required, the *Recovery Manager* will be invoked for compensating activities (cf. [26]).
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 propagates to its parent for further handling. Though a failure may propagate up the activity composition hierarchy, this approach localizes exceptions and thus reduces loss of work done.

3.5. Sample Normal Execution Sequence

In order to demonstrate how the ADOME-WFMS framework functions under normal execution conditions, Table 1 shows the normal execution sequence of the activities and tasks involved in the requisition workflow. This serves to explain the details that happen during normal execution, including the mechanisms of event-driven execution, activity composition hierarchy, match making, web interface and data flow.

3.6. Other ADOME Web-Interface Functions

Besides the functions related to activity execution, exception handling and workflow evolution, ADOME-WFMS has the support of other standard functions. Due to space limit, these standard functions of ADOME-WFMS are summarized in Table 2.

4. 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 workflow of requisition procedure in Figure 1. The composition hierarchy view showing rules and input / output of tasks is depicted in Figure 4. Then in the next section, we discuss in depth how reuse is facilitated with advanced exception modeling based on the ADOME mechanisms.

4.1. Identifying and Executing Exception Handlers

This section discusses how exception handlers are identified and executed in ADOME-WFMS with the example workflow of requisition procedure in Figure 1. The composition hierarchy view showing rules and input / output of tasks is depicted in Figure 4. Issues relating to cascaded exceptions will be highlighted later in Section 4.2. One or more exception handlers may be qualified to handle an exception that occurs. As illustrated Figure 9 (cf. Appendix A for the detailed algorithm), the ADOME-WFMS *Exception Manager* employs the following priority order for selecting the appropriate exception handler(s).

1. Mandatory ECA Handlers Since the users specify these as mandatory, all relevant handlers (with event matched and condition fulfilled) bound 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 (*ral* in Figure 4(d)) while a global mandatory ECA rule specifies all PC failures should be reported to the EDP manager (*rl* in Figure 4(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, such as logging and notification, may or may not solve the problem causing the exception. If they cannot solve the problem, other categories of exception handlers will be executed. For example, all exceptions in the 'Payment arrangement' activity (cf. Figure 1(c)) are regarded as having severe consequences, so the financial manager will be informed (*rb1* in Figure 4(d)) and manual handling of the exception is required.

<i>Activity/ Task</i>	<i>Detailed Steps during Normal Execution</i>
1. Requisition	User John requests to start "Requisition" from a web page. Web server API triggers start-event for "Requisition". Activity Executor creates a new "Purchase Request" activity instance and starts it. Activity Executor invokes the Recovery Manager to do house-keeping. Activity Executor raises <i>start</i> -event for "Purchase Request", which is the first sub-activity.
1.1 Purchase Request	Activity Executor creates a new "Purchase Request" activity instance and starts it. Activity Executor invokes the Recovery Manager to do house-keeping. Activity Executor raises <i>start</i> -event for "Get Product Information", which is the first task.
1.1.1 Get Product Information	Activity Executor creates a new "Get Production Information" task instance and starts it. Activity Executor invokes the Recovery Manager to do house-keeping. Match Maker determines John is suitable for the task. Since the PSA is the same as the one who started the task, the Web-interface replies with the interface page for this task, instead of indirectly with an ICQ message. John clicks the "start work" button because off-line work is necessary (e.g. browsing the web or reading magazines). After getting the information, John fills in the Product Information Form provided with the interface page. (He may come back later and return to this page by checking his work in progress through the web pages for management.) John clicks the "Finish" button to raise the <i>finish</i> -event of this task.
1.1.2 Fill in PR Form	Activity Executor raises the <i>start</i> -event for "Fill in PR", which is the next task. Activity Executor creates a new "Fill in PR Form" task instance and starts it. Activity Executor invokes the Recovery Manager to do house-keeping. Match Maker determines John is suitable for the task. Since the PSA is the same as the one of the last task, the Web-interface replies with the interface page for this task, instead of indirectly with an ICQ message. John fills in the Purchase Request Form provided with the interface page. John clicks the "Finish" button to raise the <i>finish</i> -event of this task. (John can monitor the progress of the rest of the requisition activity through ADOME-WFMS management pages because he is the starter.)
1.1.3 Budget Check	Activity Executor raises the <i>start</i> -event for "Budget Check", which is the next task. Activity Executor creates a new "Budget Check" task instance and starts it. Activity Executor invokes the Recovery Manager to do house-keeping. Match Maker determines a stored computer procedure suitable for the task. Activity Executor passes the Requisition Form as parameter to the computer procedure and starts it. The computer procedure verifies that budget is available with the necessary computations. The computer procedure raises the <i>finish</i> -event of this task.
1.1.4 PR Approval	Activity Executor raises the <i>start</i> -event for "PR Approval", which is the next task. Activity Executor creates a new "PR Approval" task instance and starts it. Activity Executor invokes the Recovery Manager to do house-keeping. Match Maker determines the Financial Controller is suitable for the task. The Web-interface sends the Financial Controller an ICQ message, with the URL of the interface page for this task. The Financial Controller logs on to the interface page and browses through the Purchase Request Form (and optionally the production information). The Financial Controller approves the task and clicks the "Finish" button to raise the <i>finish</i> -event of this task.
1.1.5 Purchase Request END	Activity Executor reaches to the "END" node of the "Purchase Request" activity, and raises the <i>finish</i> -event for it. Activity Executor raises the <i>start</i> -event for "Procurement" activity, with is the successor.
1.2 Procurement	Activity Executor creates a new "Procurement" activity instance and starts it. Activity Executor invokes the Recovery Manager to do house-keeping. Activity Executor raises the <i>start</i> -event for "Find Supplier", which is the first task.
1.2.1 Find Supplier	Activity Executor creates a new "Find Supplier" task instance and starts it. Activity Executor invokes the Recovery Manager to do house-keeping. Match Maker determines procurement officer Peter is suitable for the task. The Web-interface sends Peter an ICQ message, with the URL of the interface page for this task. Peter logs on to the interface page and clicks the "start work" button because off-line work is necessary Peter browses through the Purchase Request Form (and optionally the production information) and tries to find (more) suppliers for the items. Peter clicks the "Finish" button to raise the <i>finish</i> -event of this task.
1.2.2	...
...	
1.2.7 Procurement END	Activity Executor reaches the "END" node of the "Procurement" activity, and raises the <i>finish</i> -event for it. Activity Executor determines the next activity: "Payment Arrangement" if the purchase is on C.O.D. terms; otherwise "Check and Receive Goods" on credit terms. Activity Executor raises the <i>start</i> -event for the chosen activity.
...	
1.7 Requisition END	Activity Executor reaches to the "END" node of the "Requisition" activity, and raises the <i>finish</i> -event for it. John, who started this activity, receives a notification.

Table 1: Detailed Steps Illustrating Normal Execution Sequence

Activity / Task related	Search for activity / task definition (class) - by name, owner, owner's department Browse activity structure Browse activity / task requirement: deadline, PSA capability, budget, etc. Browse ECA rules / constraints attached to activities Start new activity
Activity / Task Instance Management	Search for activity / task instances (showing affected instances by workflow evolution, if any) Monitor activity / task status – progress of activities Exceptions occurred and resolutions employed
PSA related	Search for PSA by name, organization unit, capability Update PSA attributes, add / delete PSA Trial matching of task with PSA Monitor PSA status (personal task list) Capability definition management
ADOME related	Queries and updates for ADOME objects and role ADOME class / role definitions

Table 2: Other Web-Based Functions for ADOME-WFMS

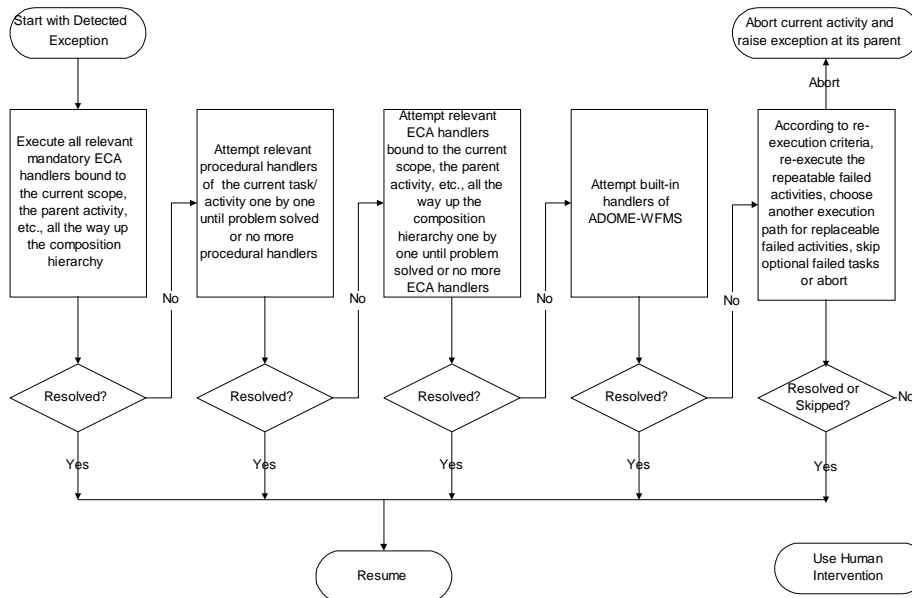


Fig. 9: Flowchart for Identifying and Executing Exception Handlers

Furthermore, the organization manager may later find that it is useful for all exceptions in a department to be reported to its manager. So rules *ral* and *rb1* are combined 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. Figure 1(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 the more specific exception type would be chosen over the more general exception type (as explained in Section 5).

For example, the rule “If financial controller is not available, the department manager can approve any task involving less than \$2,000” (*r2* in Figure 4(d)) will enable the department manager 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 Figure 4(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. 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.

5. Re-Execution Criteria If none of the above handlers is specified, ADOME-WFMS will attempt re-execution criteria. Section 4.2 discusses some measures to prevent cascaded exceptions and loops, mainly related to deadline and budget constraint. ADOME-WFMS automatically re-executes the *repeatable* failed activities; choose another execution path for *replaceable* failed activities; skip *optional* failed tasks. This feature can save many tedious explicit jumps and aborts, especially with scoping in ADOME-WFMS (cf. Section 5). Moreover, this way can resolve many unexpected exceptions if re-execution helps.

However, *critical* failed tasks without explicit handlers are unexpected exceptions. Therefore, it will result in human intervention. Upon re-execution, in order to maintain work continuity and save start 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,

- (1) The “Requisition” activity specified in Figure 4(a) is repeatable and thus the sub-activities “Purchase Request”, “Procurement”, etc., are all *repeatable* unless otherwise stated.
- (2) 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 4(a)) are *replaceable*.
- (3) Upon “Purchase Request”, the user may not need to execute the task “Get Product Information” because he may know that product 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. Figure 4(b)).
- (4) As illustrated in Figure 4(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.

4.2. Handling Cascaded Exceptions and Loops

To handle cascaded exceptions effectively and avoid infinite generation of cascaded exceptions, the following *safety* measures are employed in ADOME-WFMS: (1) Notification within the exception handling activity to the human deciding on the handler for an unexpected exception (e.g. to report even expected exceptions); (2) if cascaded exceptions occur, the same human should be notified for better management and decisions; (3) 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; (4) 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; (5) The *Activity Executor* will keep track of the sequence of executed tasks, so that 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; and (6) 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.

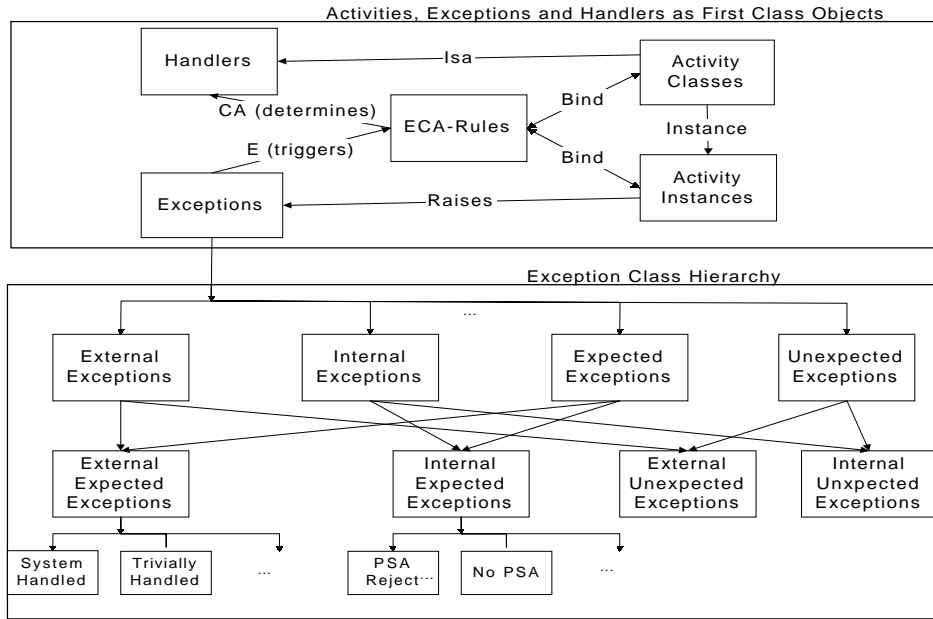


Fig. 10: Activities, Exceptions, and Handlers as First-Class Objects

5. EXCEPTION MODELING FOR REUSE IN ADOME-WFMS

Figure 10 illustrates the main entities and relationships in ADOME-WFMS with respect to exception handling. These entities are all modeled as first-class objects. In particular, the class *exceptions* is a subclass of class *events*. Figure 10 illustrates the meta-class definition for exceptions. A taxonomy of exceptions and handlers is found in [22]. Handlers are allowed to be sub-activities so that they can carry out any complicated actions; and nested exceptions are supported by recursive invocation 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, both dynamically at run-time (with *bind* statements) and at definition time. This is because the underlying ADOME active OODBMS supports ECA rules and the above-mentioned dynamic features [15, 17, 18]. 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. This is because handlers can be inherited down the activity composition hierarchy. Section 5.2 illustrates the rules bound to the requisition activity and its sub-activities at definition time.

It should be noted that the *Exception Manager* can handle multiple exceptions from different tasks and activities concurrently. For multiple exceptions to be handled together in as a single one, composite exceptions in the form of composition events can be defined and thus exception handlers accordingly.

The ADOME-WFMS *Human Intervention Manager* supports users to modify all the above declarations and associations at run-time as described in Section 6. The power of ADOME-WFMS in reuse over other systems (such as [8, 29, 34, 39, 41, 51]) is mainly due to ADOME's ability in dynamic binding of rules to different dimensions (objects, roles, sub-activities, etc.) at run-time as explained in Section 5. However, a methodology in workflow design to facilitate reuse of exception handlers is beyond the scope of this thesis.

However, a methodology in workflow design to facilitate reuse of exception handlers is beyond the scope of this article.

5.1. Reusing Workflow Definitions with Activity Composition Hierarchy

The activity composition hierarchy specifies and records the activities and their constituting sub-activities recursively down to the task level. Tasks are atomic activities with no sub-activities. Sub-

activities are made up of tasks. Once the user defines an activity, this decomposition is stored and made available for reuse. Thus, a sub-activity may be part of many other activities. Users can easily compose complicated activities from existing activities/sub-activities/tasks. On the other hand, if the user decides that a task should be refined (say, for better progress monitoring, capturing intermediate results, change of complexity or requirements, etc.), ADOME-WFMS supports the operation of exploding a task into a complex sub-activity. For example, the task 'receive-and-check-goods' may be refined to a sub-activity composed of tasks 'check-quantity', 'check-quality' and 'sign-delivery-note' so that each step can be carried out by different agents. In this way, ADOME-WFMS allows the design of workflows in both top-down and bottom-up manners.

On the other hand, the activity composition hierarchy is important for encapsulating details of activities and sub-activities so as to facilitate: (a) reuse of task and sub-activity definition in other new activities; (b) stepwise refinement by decomposing an activity into sub-activities representing more elementary tasks if required; (c) easy maintenance for activity definition and exception handler definitions; (d) scoping for exception handlers; (e) the use of nested transaction models for execution control; (f) localizing failures and thus limiting the loss of work done; (g) capturing exceptions during task execution; (h) reuse and sharing of ECA rules and meta-ECA rules

5.2. 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, some mechanisms for reuse of exception handlers follow from its hierarchy activity structure. For procedural exception handlers, arcs from several 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). For example, declaring *r2* at the *requisition* activity level will enable *r2* for all the sub-activities and tasks for the whole diagram.

Similarly, human intervention requirements of exception handling (automatic, warning, system-assisted 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.

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. For example, we can declare $r9=(E:All_exception, amount>1000000, CEO.inform)$ and then bind *r9* to *payment*, *requisition*, so that all exceptions in the payment and requisition sub-activities will inform the chief executive officer for transactions greater than one million dollars.

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.

Moreover, meta-level rules can be instantiated through parameters and supplied methods to specify rules, such as budget rules instantiated with actual budget figures.

5.3. 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 them. 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.

```

PROCEDURE Human_Intervention(ex: Exception; act: Activity)

  p = FindPSA(act.human_intervention)

  REPEAT
    Compute a list P1 of alternate paths (if any)
    Compute a list L1 of all mandatory ECA handlers (if any)
    Compute a list L2 of all procedure handlers (if any)
    Compute a list L3 of all ECA handlers (if any)
    Compute a list L4 of internal (WFMS) handler (if any)
    Construct a list R1 of possible resolutions by maintaining execution
      behavior. (according to [23])
    Construct a list R2 of possible resolutions by modifying execution
      behavior. (according to [23])
    Construct a list R3 of possible resolutions by workflow evolution.
      (according to Section 6.4,[23])

    Display the above lists with a structured menu to the selected PSA p.
    Attempt to provide recommendation according to the Table 3 below.
    Ask p to select additional resolution action(s) res (if any) from R1,R2,R3.
    FOREACH r IN res DO BEGIN
      Ask p whether r is Temporary, Immediate, Delayed or Abrupt
      Verify r according to Table 4 below.
      Execute r
    END
    Ask p to select exception handler(s) found in L1-L4 (if any).
  UNTIL ex.Check=FALSE OR (p wants to stop trying)

  Ask p the next activity: redo, skip, jump, abort or alternate paths P1.
  Execute the next selected activity.

END

```

Fig. 11: Main Algorithm for ADOME-WFMS Human Intervention

Reuse is supported through inheritance down the organization composition hierarchy for organization, units, sub-units and groups. For example, 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 if they do not violate that of the organization. Moreover, reuse is supported through any other class and sub-classes of objects in ADOME-WFMS (such as PSA and resources).

Reuse is supported through inheritance down the Activity Composition Hierarchy for projects (any collection of activities), activities, sub-activities, tasks.. 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.

Reuse is supported through multiple-inheritance down the Role/Token Hierarchy for positions and capability. 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[16].

Finally, reuse is supported through combination of the above levels and dimensions. As supported by ADOME [16], 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.

6. WEB-BASED HUMAN INTERVENTION MANAGER

In this section, we detail the ADOME-WFMS Human Intervention Manager, including a workflow evolution approach. Through its web-based facilities, we illustrate how unexpected exceptions are handled in ADOME-WFMS, and how reuse can be facilitated. Figure 11 shows the main algorithm of the Human Intervention Manager.

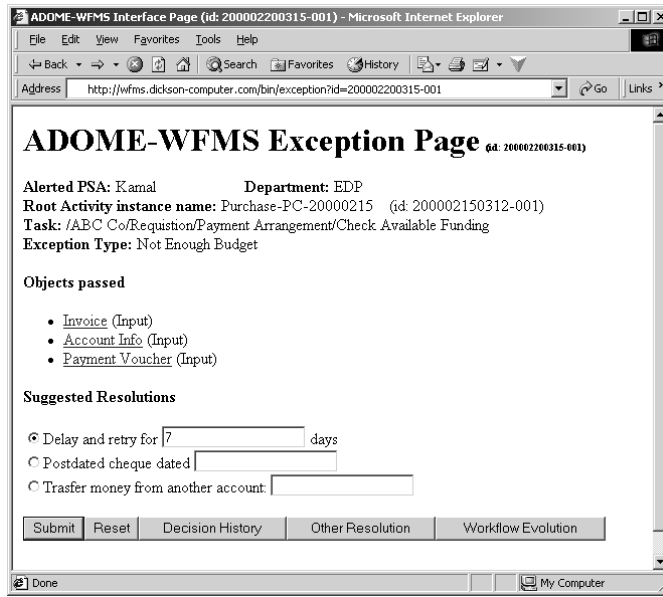


Fig. 12: ADOME-WFMS Exception Page

<i>Situations</i>	<i>Suggested resolutions</i>
Cannot find PSA	Wait, skip / postpone task, change requirement for a task, amend capabilities for PSA, abort other tasks to release PSA, switch PSA assignment, add PSA
Not enough resources	Wait, skip / postpone task, change resources requirement for a task, add resources, abort other tasks to release resources
Cannot meet deadline	Postpone deadline, add PSA, expedite activity execution
Activity constraint violation	Change constraint (e.g. budget), change relevant data objects
Task fail	Skip / postpone / repeat task, alternate branches of execution, change PSA / resources requirement
Activity fail	Abort current sub-activity, alternate branches for execution

Table 3: Suggested Resolution for Some Workflow Situations

6.1. Handling Unexpected Exceptions

Upon *unexpected exceptions* or *manual* exception handling is specified, the *Human Intervention Manager* sub-module of the *Exception Manager* will alert the specified user by ICQ (or if not successful, then by electronic mail), with the URL of an *Exception Page* (cf. Figure 12) that assists the user to handle the exception. This web page, again generated from a template, offers reuse of existing exception handlers by providing a list of possible resolutions, and relevant data objects to assist the human decision. 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 are the most appropriate actions. Table 3 summarizes the suggested resolution for some exception cases under different situations[†].

On the other hand, especially when *manual* exception handling is specified, there may exist a list of more concrete exception resolutions, which probably needs the human user to choose since there may not be enough knowledge/experience for the system to totally automate such a decision. For example, when the task “check available funding” in Figure 1(c) fails, the exception would trigger *manual* handling as shown in Figure 12. Here, the user can choose from a list of *suggested resolutions* (with parameters) or browse/edit for other resolutions (such as returning the goods). Before making a decision, the user may browse the *objects passed* for further information or consult the *Decision History*.

[†]A comprehensive system assisting such decision-making is beyond the scope of this paper.

<i>Exception Handling Action</i>	<i>Verification</i>
Changing capability / resources requirement	Budget, availability of resources
Adding PSA or other resources	Budget, deadline (e.g., hiring workers and purchasing takes time)
Waiting	Deadline and other temporal constraints
Repeating task / alternate paths	PSA / resources availability, budget, deadline
Aborting other tasks for resources / PSA	Check importance of victim task, time and cost implications

Table 4: Sample Verifications for Exception Handling Actions

6.2. Verification for Human Decision

Prior to executing the action specified through human intervention, the Exception Manager performs checking to avoid further cascaded exceptions. In addition to generally specified constraints in the ADOME-WFMS, some of its specific ones are illustrated in Table 4. Should some constraints be violated, the Exception Manager will further inform the user of the potential problem and ask for confirmation. There are often scenarios where one may handle exceptions for a more important task at the expense of sacrificing other less important ones.

6.3. Exception Handling Example

To demonstrate how the ADOME-WFMS framework handles expected and unexpected exceptions, Table 5 illustrates some of the exceptions in the example requisition workflow and their handling resolutions under ADOME-WFMS.

6.4. Web-Based Workflow Evolution

Typical workflow evolution methods are illustrated in Table 6. These are implemented as library methods so that users can easily choose their desired workflow evolution. Upon pressing the *workflow evolution* button, the user can access a workflow evolution menu page as shown Figure 13, with a set of possible workflow evolution options. However, a full-functioned graphical interface for workflow evolution is under construction and is beyond the scope of this article.

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 are 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.

As an example, after running the procurement example workflow (version 1) of Figure 1 for some time, the administrator observes that exceptions in ‘user evaluation’ and ‘purchase approval’ occur too frequently for ad hoc manual handling. Thus, the ‘Procurement’ activity is changed to version 2 (as detailed in Figure 14) by adding additional arcs and tasks for exception handling. (An *immediate change* is possible in this case since extra enhancements are added.) These changes may be due to experience gained from previous user interventions, where the users chose to jump back to the appropriate position to restart (*temporary changes*).

<i>Activity/ Task</i>	<i>Exception Condition</i>	<i>Exception Handling Resolution</i>
1.1 Purchase Request	The critical component tasks "PR approval" or "Budget Check" may fail and abort this activity.	The activity's re-execution mode is repeatable, but there is no specific handler. The activity is executed again.
1.1.1 Get Product Information	The user is unable to find product information needed and raises "no_prod_info".	The task's re-execution mode is optional and there is no specific handler. This exception is simply ignored (This is not important because the procurement office can do that later when suppliers are searched for.)
1.1.3 Budget Check	The software that checks budget raises "Budget_Exceeded".	The task's re-execution mode is critical and there is no specific handler. The whole "Purchase Request" activity fails.
1.1.4 PR Approval	The Financial Controller rejects being a PSA for the task	The handler r2 specifies the department manager to be the PSA if the amount of money involved is less than \$2000. (The Financial Controller may have rejected the invitation on purpose so as to let the department manager decide.) Subsequently, if the department manager is also not available (time out while waiting), this is a cascaded exception. The task's re-execution mode is critical and there is no specific handler. The whole "Purchase Request" activity fails
1.1.4 PR Approval	The Financial Controller raises "PR_not_approved".	The task's re-execution mode is critical and there is no specific handler. The whole "Purchase Request" activity fails.
1.2 Procurement	The "Find Supplier" task fails this activity with "Supplier not found".	The activity's re-execution mode is repeatable, but there is no specific handler. The "Procurement" activity is executed again.
1.2 Procurement	The "Supplier Confirmation" task fails this activity with "Reject_order".	The procedure handler (arc) at the "Requisition" activity level leads the execution back to "Purchase Request" step (so that the user can refine his purchase request).
1.2.1 Find Supplier	The procurement officer raises "Supplier_Not_Found".	The procedure handler (arc) leads to the end of the activity with failure.
1.2.2 Get Quotation & Prod. Info.	The procurement finds the user's budget too low.	Since manual mode is specified for exception handling, the procurement officer handles the exception by trying to find other suppliers, send back the request to the user for revision, or some other ad-hoc ways.
1.2.3 User Evaluation	The user is unhappy with the supplier because of bad records.	Since manual mode is specified for exception handling, the user handles the exception by trying to find other suppliers, asking the procurement officer to find other suppliers, or by some other ad-hoc ways.
1.2.4 Purchase Approval	The Purchasing Manager raises "Too_Expensive".	Since manual mode is specified for exception handling, the user handles the exception by either trying to find other suppliers, asking the procurement officer to find other suppliers, abort the "Procurement" activity or by some other ad-hoc ways.
1.2.6 Supplier Confirmation	The Supplier raises "Reject_order".	The task's re-execution mode is critical and there is no specific handler. The whole "Procurement" activity fails.
1.3 Payment Arrangement	Payment arrangement for C.O.D. fails	Since there is another replaceable branch, the activity executor would attempt the "Credit" branch instead.
1.3.1 Match PR, PO and Invoice	Invoice is lost.	Since manual mode is specified for exception handling, the accounting officer handles the exception manually. However the mandatory handler r3 helps in sending an email to the supplier to ask for re-issuing the invoice automatically. At the same time, due to mandatory handler rb1, the Finance Manager is informed.
1.3.2 Check Available Funding	Funding is not available to pay for the invoice.	Since manual mode is specified for exception handling, the accounting officer handles the exception manually (as shown in Figure 12). At the same time, due to mandatory handler rb1, the Finance Manager is informed.
1.4 Check & Receive Goods	The quality assurance officer raises "Poor Quality".	Handler rule rd1 will pass the case to the purchasing manager for manual follow-up actions. The purchasing manager may decide to cancel the order and return the goods received, which is a cascaded exception. In this case, compensation activities are also required (cf. Chapter 6).

Table 5: Sample Exception and Handling Resolutions

<i>Workflow Evolution Action</i>	<i>ADOME-WFMS Method Specification</i>
Change PSA capability	PSA.Set_cap(s: set of Token)
Add sub-activity which recruits for more PSA	Add_PSA.start(capability_required: set of Token; b: budget; deadline: time)
Add sub-activity which obtains more resources	Add_Resource.start(resources_required: set of resources; b: budget; deadline: time)
Change capability requirement	Task.Set_cap_req(t: Token, m: Change_mode)
Change resource requirement	Task.Add_res_req(r: Resource, m: Change_mode) Task.Delete_res_req(r: Resource, m: Change_mode)
Change constraints of activity	Activity.Add_constraint(r: ECA_rule, m: Change_mode) Activity.Delete_constraint(r: ECA_rule, m: Change_mode)
Changing pre-conditions for transitions	Activity.Set_precond(c: Condition, m: change_mode)
Delete branch arc	Activity.Del_branch(l: Label, m: change_mode)
Add branch arc	Activity.Add_branch(destination: Activity; c: Condition, l: Label, m: change_mode)
Insert (preparation) work step	Activity.Add_pre_step(work: Activity, l: Label, m: change_mode)
Add procedural handler	Activity.Add_proc_handler(work: Activity, c: Condition, l: Label, m: change_mode)
Add ECA rule (handler)	Activity.Add_rule(r: ECA_rule, m: change_mode)

Table 6: Sample Specification for Workflow Evolutions in ADOME-WFMS

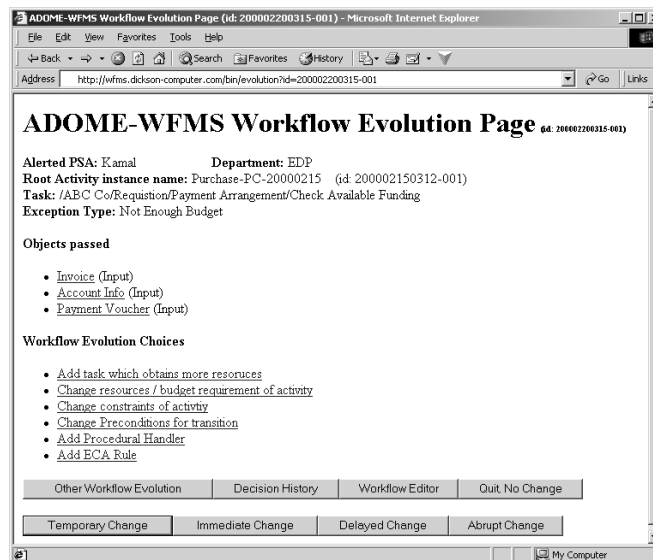


Fig. 13: ADOME-WFMS Workflow Evolution Page

In addition, if many suppliers complain about late payment and refuse to deliver ordered goods, the ‘credit’ branch of Figure 1(a) may be completely deleted. Thus, instead of delayed 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 C.O.D. payment in this case will be just the date on the cheque and the payment procedure becomes much simplified.) These are workflow evolution examples, which can support possible exception handling in ADOME-WFMS; such workflow evolution may lead toward (eventual) exception avoidance.

6.5. Discussion

To maintain the consistency of the evolved workflow, we use similar techniques as elaborated in related work [11, 46]. Similarly, the resolutions for human intervention can be application domain specific and based on the skill level of the users. Thus, ADOME-WFMS facilitates a customizable application domain specific support for resolutions based on human interaction. Finally, consistency

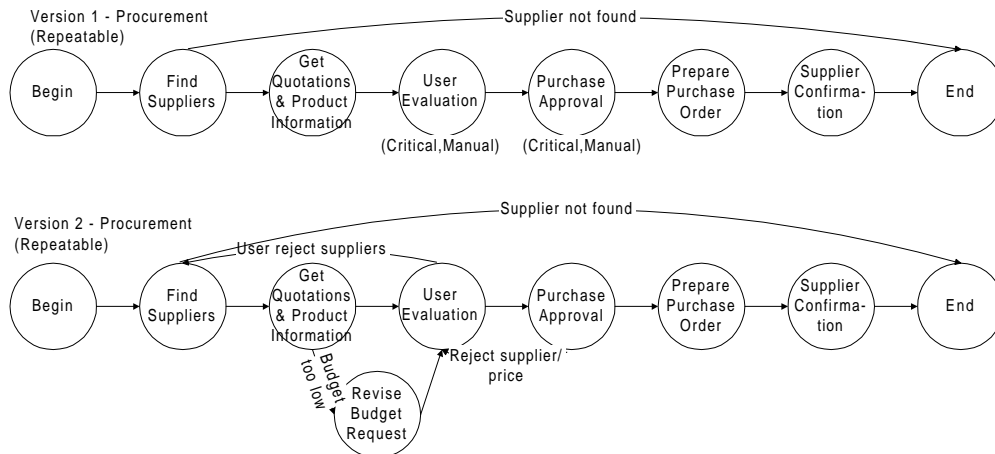


Fig. 14: Example of Workflow Evolution (Continuation of Figure 1)

checks shown in Table 4 can be employed to check whether decisions taken by humans are appropriate. Therefore, ADOME-WFMS provides a comprehensive framework for exception handling and builds on our meta-modeling approach to exception handling [22], and comprehensive set of exception handlers dealing with both automatic and manual resolution.

7. RELATED WORK

In this session, we review related work. It is a new approach to E-service enactment based on an advanced WFMS engine. Besides E-ADOME, other notable systems using related approaches include Eflow [14] and Crossflow [33], which is also mentioned in this session.

7.1. Web Interface for WFMS

Dartflow [6] is one of the first web-based WFMS, using transportable agents, CGI and Java technologies. WebWork [48, 49] described some issues in web-based workflow recovery, but only on WFMS and web related failures and not covering user-defined workflow exceptions. WW-flow [42] provides a hierarchical control scheme over activities implemented in Java for both the workflow engine and client interfaces. It allows sub-activities to be executed in different workflow engines across the web. Eflow [14] is one of the closest commercial systems with features like E-ADOME [22] in handling e-Services. However, Eflow does not address matching of agents directly with tasks. Instead, it uses the concept of generic service node and service selection rules. Currently, several commercial WFMSs such as TIB/InConcert [54] and Staffware 2000 [52], provide web user interface too. In addition, I-Flow [32] has a Java workflow engine.

However, not all the above-mentioned WFMSs support for web-based cooperative exception handling. Most of them contact clients based on electronic mail and web forms and does not directly support active paging of clients with Internet message facilities like ICQ. As for standards, Workflow Management Coalition (WfMC) has recently proposed Wf-XML [56], which is an interchange format specification for an XML language designed to model the data transfer requirements for process specification.

7.2. Flexibility of Objects and Schema

Flexibility of objects and schema facilitates exception handling since real-time modifications to objects, roles, rules and workflow are required. In particular, a common convenient way employed by many contemporary WFMS to support all these features in a unified platform is to use active OODBMS technologies. A detailed survey on active OODBMS, comparing HiPAC, Sentinel, ADAM, Chimera,

NAOS, ODE, SAMOS, and ADOME can be found in [19]. The main feature in an active OODBMS that provides support for various exception-handling features is the ability to dynamically bind rules to classes, objects and roles. This enables exception-handling rules to be associated with different activities and tasks, thus facilitating reuse, in addition to standard OO inheritance and composition.

The ability of schema evolution is required for workflow evolution, since workflows are defined as classes in the OO paradigm. (In relational approaches, such as in [41], changing a workflow definition requires changing data items in tables, instead of table definitions, because workflow definitions are stored in tables.) The nature of the OO model permits the semantics of schema evolution to be rigidly defined and validated [35]. Early work on schema evolution in OODBMS can be dated back to [3] where taxonomy of primitive schema evolution operations was defined for the ORION OODBMS. Itsaca [37], the OODBMS on which ADOME based, supports a similar set of features and thus facilitates workflow evolution in ADOME-WFMS.

7.3. Workflow Evolution

One of the earliest work in workflow evolution is [31], where correctness criteria for instance migration are defined based on the definition of the set of all valid node sequences, i.e., a change is valid if the execution sequence can be obtained with the new workflow definition. However, this article describes a limited set of change primitives.

[13] described a set of primitives for migrating workflow schema and instances. [46] described how individual workflow instances could be handed over to the newly migrated workflow schema by using rules. We apply the techniques of these two research groups in designing ADOME-WFMS workflow evolution primitives and operations. ADEPTflex [51] developed a set of change operations, which support users in modifying the structure of running workflow instances. Changes are controlled against undesirable consequences using dataflow constraints expressed in the workflow. Constraint violations will cause the change to be rejected or further remedy exception handling (possibly leading to concomitant changes).

PROSYT [28] addressed inconsistencies and deviations in general process support systems (where WFMS is considered as a kind of process support system), but the contribution was more on the formal modeling than semantic modeling. They support change of execution path for a workflow instance only. [5] presented a unified framework for tolerating exceptions for data and processes in WFMS, but without details at the logical and implementation levels. They addressed some problems for offending instances and not for schema or workflow evolution.

Most related work in workflow evolution addresses exceptions caused by inappropriate workflow evolution rather than how workflow evolution can contribute to exception handling and avoidance (cf. Section 6.4).

7.4. Exceptions in Related Environments

Sophisticated programming language features for exception handling date back to the Ada programming language [1]. The feature of exception propagation (i.e. trace back the execution chain of calling procedures for an appropriate handler) for resolving exception handlers influences the ADOME-WFMS exception manager. Moreover, Ada's generic procedures, and thus generic exception handlers, motivate the employment of meta-modelling of exceptions and handlers for ADOME-WFMS.

[4] is a classical article on exception handling but for general database-intensive information system. Exceptions are treated as classes of objects, but the work is based on constraints and not the ECA paradigm. [52] built a taxonomy and suggested a meta-model on exception handling for office information systems, with handling methods for different kinds of exceptions and suggests a model employing a set of unique input/output (UIO) sequences for verifying exception handling rules. [6] presents a framework for a comprehensive and extensible exception handling mechanism for a concurrent object-oriented environment, including language constructs and various exception features.

HiPAC [19] represents a classic active OODBMS. Many of its features and concepts, such as treating rules and events as objects, ECA rules, composite events, exceptions as events, etc., influence the design of many later systems.

7.5. Exception Handling in Commercial WFMS

Very few commercial WFMSs provide support for handling exceptions. Even if they do, they only address very basic problems to a slight extent.

InConcert, by InConcert Inc. [47], supports event-action triggers. Triggering events can be process state changes, external user-defined events or temporal events. Actions can be notification messages to agents, executing another process, or calling a user-defined procedure. Staffware, by Staffware Corporation [53], and Changengine, by Hewlett-Packard [36] support special tasks called event nodes, which can suspend the execution of a workflow instance on a give path until a pre-defined (exception) event occurs, and then can execute an event handling sub-activity in the workflow. [13] discusses how various kinds of expected exceptions can be mapped on top of Changengine.

7.6. Some Other OO WFMS Framework

The Workflow Management Coalition (WfMC) is working towards an industrial standard with the WfMC Reference Model [57] for WFMS so as to identify their characteristics, terminology [58] and components, and to enable the individual specification to be developed within the context of an overall model for WFMS. However, only very recently, WfMC published a white paper on event extension on WFMS [59], but they still do not specifically address exceptions.

Among these WFMS, TriGSflow [39], one of the earlier OO WFMS, is perhaps the closest system to ours in that it adopts an OO design, and utilizes rules and roles. However, TriGSflow only uses roles for associating agents with tasks, but not for modeling capabilities needed in matching agents with tasks. Furthermore, it has little support for exception handling. [44] described a framework of using roles, event-based execution model and exception handling rules in a WFMS. However, they do not provide an organization model nor address a variety of exception conditions. Further, they do not support activity decomposition or capability matching to facilitate workflow definition and execution.

[43] adopted a knowledge-base approach to handle expected exceptions in WFMS in the form of a “process handbook” with strong emphasis on agent management. OPERA [34] incorporates primitives based on exception handling concepts developed for programming languages coupled with ideas from advanced transaction models. It is a parallel work, which arrives at a similar basic exception handler resolution strategy as ADOME-WFMS, but from transactional workflow point of view. OPERA models exception handlers as sub-activities and supports exception propagation to the parent activity if the exception is not handled in the child activity.

WIDE [8] used object and rule modeling techniques and suggested some measures in handling exceptions. In [11], exceptions were classified but not handling approaches. [10] addressed reuse and management of exception handlers with implementation details, but not adequately considered high level semantics. [9] used workflow evolution primitives, but again not at a semantic level. The main drawback of WIDE is missing facilities for coordination of agents and also lacks a coherent model of various entities and their inter-relationship in a WFMS. Their literature is more on theoretical aspects rather than a logical framework showing how their concepts can be implemented.

Crossflow [33] models virtual enterprises based on a service provider-consumer paradigm, in which organizations (service consumers) can delegate tasks in their workflows to other organizations (service providers). High-level support is obtained by abstracting services and offering advanced cooperation support. Virtual organizations are dynamically formed by contract-based matchmaking between service providers and consumers.

On the other hand, CapBasED-AMS [41] is implemented on a relational system with active capability. The modeling for workflow is not straightforward since almost all designs (which are in essence object-oriented) need to be converted into the traditional relational model. Moreover, the system can weakly support features that require more advanced features such as dynamic schema modification – a basic requirement for workflow evolution. However, the notion of event-driven execution and capability reasoning in ADOME-WFMS has been influence by CapBasED-AMS.

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 agents, match-making, exception handling, etc.). Compared with the systems close to us, ADOME-WFMS has the most features and can support mobile agents in the Internet.

8. CONCLUSION

This article has presented adaptive exception handling in ADOME-WFMS, a flexible WFMS based on an active OODBMS extended with role and rule facilities (viz., ADOME). Compared with other research on this topic, ADOME provides an advanced object-oriented 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 are also possible. This article has also described in detail, how expected exceptions are actually resolved with the ADOME-WFMS Exception Manager and how unexpected exceptions are handled through its *Web Interface*. 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. Moreover, the *Web Interface* also demonstrates effective management of human PSAs, especially during exceptions.

ADOME-WFMS is currently being built on top of the ADOME prototype system, with a web-based user interface to accommodate the whole range of activities. We are also porting the ADOME-WFMS *Activity Editor* to a web-based version to support web-based workflow evolution with full graphics capability. On the theoretical side, we are looking at reuse models in a more formal way, and the possibility of applying UML technologies to this aspect. We are also interested in performance evaluation and optimization of workflow exception handling [39].

Further research issues on interfacing and interoperability are emphasized for extending the applicability of an advanced WFMS engine, which includes: expanding the possible interfaces and coordinating different types of agents, graphical workflow evolution tools, and inter-operating with other WFMS. We are in the process of developing E-ADOME [22], an extension of ADOME-WFMS with coordination mechanisms for different kinds of agents, in various advanced real-life e-commerce environments, such as procurement, finance, stock-trading and insurance. We are especially interested in the RosettaNet Internet business communications standards (a framework using XML to streamline the computer product and IT supply chain).

Finally, it should be noted that exceptional handling, and especially unexpected exception handling is a very tough problem which requires an in depth application domain knowledge to build a usable solution. The solutions elaborated in this paper can be fine-tuned for a specific application domain or area (as illustrated in [22]) so that such usable and practical systems can be built. In this regard, this paper describes in detail such a pragmatic solution to the exception-handling problem in workflow management systems.

Acknowledgements — This research has been funded by HKSAR RGC 747/96E and RGC 6098/99.

REFERENCES

- [1] Ada 95 Language Reference Manual (LRM) - the revised international standard. (ISO/IEC 8652:1995): *Information Technology -- Programming Languages -- Ada*, International Standards Organization (1995).
- [2] G. Alonso, R. Günthör, M. Kamath, D. Agrawal, A. El Abbadi, and C. Mohan. Exotica/FMDC: a workflow management system for mobile and disconnected clients. *Distributed & Parallel Databases*, 4(3):229-247 (1996).
- [3] J. Banerjee, W. Kim, H. Kim, and H. Korth. Semantics and implementation of schema evolution in object-oriented databases. *SIGMOD Conference*, San Francisco, California, pp. 311-322, ACM Press (1987).
- [4] A. Borgida. Language features for flexible handling of exceptions in information systems. *ACM Transactions on Database Systems*, 10(4):565-603 (1985).
- [5] 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, ACM Press (1999).
- [6] P. Buhr and W.Y.R. Mok. Advanced exception handling mechanisms. *IEEE Transactions on Software Engineering*, 26(9):820-836 (2000).
- [7] T. Cai, P. Gloor, and S. Nog. *DartFlow: A Workflow Management System on the Web using Transportable Agents*, Technical Report PCS-TR96-283, Dartmouth College, Hanover, N.H. (1996).

- [8] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Conceptual modelling of workflows. In *Proceedings of the International Conference on Object-Oriented and Entity-Relationship (OOER'95)*, Gold Coast, Australia, pp. 341-354, Springer-Verlag, Berlin (1995).
- [9] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. In *Proceedings of the 15th ER'96 International Conference*, Cottbus, Germany, pp. 438-455, Springer-Verlag, Berlin (1996).
- [10] F. Casati, M.G. Fugini, and I. Mirbel. An environment for designing exceptions in workflows. In *Proceedings of CAiSE '98*, Pisa, Italy, LNCS Springer Verlag (1998).
- [11] F. Casati. A discussion on approaches to handling exceptions in workflows. *CSCW Workshop on Adaptive Workflow Systems*, Seattle, WA, USA (1998).
- [12] F. Casati and G. Pozzi. Modeling exceptional behaviours in commercial workflow management systems. In *(COOPIS'99)*, Edinburgh, Scotland, pp 127-138, IEEE Press (1999).
- [13] F. Casati. *Models, Semantics, and Formal Methods for the Design of Workflows and their Exceptions*. PhD thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy (1998).
- [14] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M. Shan. *Adaptive and Dynamic Service Composition in eFlow*. HP Laboratories Technical Report HPL-2000-39 (2000).
- [15] L.C. Chan, D.K.W. Chiu, and Q. Li. *A Versatile Bridging Mechanism with a Experimental User Interface for An Expert OODBMS*. Technical Report HKUST-CS95-35, Computer Science Dept., Hong Kong University of Science and Technology (1995).
- [16] L.C. Chan. *Extending an Advanced Object Modelling Environment with Versatile Rule Sharing & Reactive Capabilities*. M.Phil. Thesis, Computer Science Dept., Hong Kong University of Science and Technology (1995).
- [17] 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, Texas A&M University (1996).
- [18] L.C. Chan and Q. Li. An extensible approach to reactive processing in an advanced object modelling environment. In *Proceedings of the 8th Intl. Conf. on Database and Expert Systems Applications (DEXA '97)*, LNCS(1308, Toulouse, France), pp. 38-47, Springer-Verlag (1997).
- [19] S. Chakravarthy. Rule management and evaluation: an active DBMS perspective. *SIGMOD Record*, **18**(3):20-28 (1989).
- [20] D.K.W. Chiu and Q. Li. A three-dimensional perspective on integrated management of rules and objects. *International Journal of Information Technology*, **3**(2):98-118 (1997).
- [21] D.K.W. Chiu. *Exception Handling in an Object-Oriented Workflow Management System*. Ph.D. Thesis, Computer Science Dept., Hong Kong University of Science and Technology (2000).
- [22] D.K.W. Chiu, K. Karlapalem, and Q. Li. E-ADOME: A framework for enacting e-services. *VLDB Workshop on Technologies for E-Services*, Cairo, Egypt (2000).
- [23] D.K.W. Chiu, Q. Li, and K. Karlapalem. A meta modeling approach for workflow management system supporting exception handling. Special issue on method engineering and metamodeling, *Information Systems*, Pergamon Press, Elsevier Science, **24**(2):159-184 (1999).
- [24] D.K.W. Chiu, Q. Li, and K. Karlapalem. A logical framework for exception handling in ADOME workflow management system. *International Conference on Advanced Information System Engineering (CAiSE'00)*, Stockholm, Sweden, Springer-Verlag (2000).
- [25] D.K.W. Chiu, Q. Li, and K. Karlapalem. Web interface-driven cooperative exception handling in ADOME workflow management system. In *Proc. Of the 1st International Conference on Web Information System Engineering (WISE'00)*, Hong Kong, pp. 174-182, IEEE Computer Society Press, (2000).
- [26] 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, **1**(3) (2000).
- [27] GHG Corp. Clips Architecture Manual, Version 5.1., (available at <http://www.ghg.net/clips/CLIPS.html>) (1992).
- [28] G. Cugola. *Inconsistencies and Deviations in Process Support Systems*. PhD thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy (1998).
- [29] A. Dogac, T. Ozsu, and A. Sheth, editors. *Proceedings of the NATO Advanced Study Institute (ASI) Workshop on Workflow Management Systems and Interoperability*, Istanbul, Turkey, Springer-Verlag (1997)
- [30] J. Eder and W. Liebhart. The workflow activity model WAMO. In *Proceedings of the 3rd International Conference on Cooperative Information Systems (CoopIS'95)*, Wien, Austria, pp. 87-98, University of Toronto (1995).
- [31] C.A. Ellis, K. Keddera, and G. Rozenberg. Dynamic change within workflow systems. In *Proceedings of the ACM Conference on Organizational Computing Systems (COOCS '95)*, Milpitas, California, pp. 10-21, ACM Press (1995).
- [32] Enix Consulting Limited. An Independent Evaluation of i-Flow Version 3.5, (available at <http://www.i-flow.com>) (2000).
- [33] P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig. CrossFlow: Cross-organizational workflow management in dynamic virtual enterprises. *International Journal of Computer Systems Science & Engineering*, **15**(5):277-290 (2000).
- [34] C. Hagen and G. Alonso. Exception handling in workflow management systems. *IEEE Transactions on Software Engineering*, **26**(10):943-957 (2000).

- [35] J. Huges. *Object-Oriented Databases*. Prentice Hall (1991).
- [36] Hewlett Packard. Changengine Admin Edition (AdminFlow) Process Design Guide (1998).
- [37] Itasca Reference Manual, Ibx Corporation (1994).
- [38] P. Weverka. Mastering Icq: The Official Guide. IDG Books. ICQ Press, <http://www.icq.com> (2000).
- [39] E. Kafeza and K. Karlapalem. Temporally constrained workflows. *11th International Conference on Database and Expert Systems Applications (DEXA-00)*, pp. 232-242, London, Springer-Verlag (2000).
- [40] G. Kappel, B. Pröll, S. Rausch-Schott, and W. Retschitzegger. TriGSflow - active object-oriented workflow management. *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS '95)*, pp. 727 - 736, [0-8186-6935-7], IEEE Press (1995).
- [41] 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*, Wien, Austria, University of Toronto, pp. 205-219 (1995).
- [42] Y. Kim, S. Kang, D. Kim, J. Bae, and K. Ju. WW-Flow: web-based workflow management with runtime encapsulation. *IEEE Internet Computing*, **4**(3):56-64 (2000).
- [43] M. Klein and C. Dellarocas. A knowledge-based approach to handling exceptions in workflow systems. *Proceedings of the Third International Conference on Autonomous Agents*, Seattle, Washington, (available at <http://ccs.mit.edu/klein/papers/aa99.ps>) (1999).
- [44] A. Kumar and J.L. Zhao. Dynamic routing and operational integrity controls in a workflow management system. *Management Science*, **45**(2):253-272 (1999).
- [45] Q. Li and F.H. Lochovsky. ADOME: an advanced object modelling environment. *IEEE Transactions on Knowledge and Data Engineering*, **10**(2):255-276 (1998).
- [46] C. Liu, M. Orlowska, and H. Li. Automating handover in dynamic workflow environments. In *Proceedings of the 10th International Conference on Advanced Information Systems Engineering CAISE'98*, Pisa, Italy, pp 159-172, Springer-Verlag (1998).
- [47] D. McCarthy and S. Sarin. Workflow and transactions in InConcert. *IEEE Data Engineering*, **16**(2):53-56 (1993).
- [48] J.A. Miller, Devanand Palaniswami, Amit P. Sheth, Krys J. Kochut, and Harvinder Singh. WebWork: METEOR2's web-based workflow management system. *Journal of Intelligent Information Systems*, Special Issue on Workflow Management Systems, Kluwer Academic Publishers, **10**(2):185-215 (1998).
- [49] J.A. Miller, A.P. Sheth, K.J. Kochut, and Z. Wei Luo. Recovery issues in web-based workflow. *Proceedings of the 12th International Conference on Computer Applications in Industry and Engineering (CAINE-99)*, Atlanta, Georgia, pp. 101-105, International Society for Computers and their Application (1999).
- [50] 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).
- [51] M. Reichert and P. Dadam. ADEPTflex - supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, **10**(2):93-129 (1998).
- [52] H. Saastamoinen. *On the Handling of Exceptions in Information Systems*. PhD thesis, University of Jyväskylä (1995).
- [53] Staffware Corporation. Staffware Global - Staffware's Opportunity to Dominate Intranet based Workflow Automation, <http://www.staffware.com> (2000).
- [54] TIBCO Software Inc., which has acquired InConcert Inc., <http://www.tibco.com> (2000).
- [55] D. Worah and A. Sheth. Transactions in transactional workflows. In S. Jajodia and L. Kerschberg, editors, *Advanced Transaction Models and Architectures*, Kluwer Academic Publishers, New York (1997).
- [56] Workflow Management Coalition. Workflow Standard – Interoperability Wf-XML Binding, WFMC-TC-1023 (2000).
- [57] Workflow Management Coalition. The Workflow Reference Model, WFMC-TC-1003, 19-Jan-95, 1.1 (1995).
- [58] Workflow Management Coalition. Terminology and Glossary, WFMC-TC-1011, Feb 1999, 3.0 (1999).
- [59] Workflow Management Coalition - David Hollingsworth, ICL A&TC. White Paper – Events (1999).
- [60] J. Widow and S. Ceri, editors. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann Pub. Inc. (1996).

APPENDIX A: DETAILED ALGORITHM FOR ADOME-WFMS EXCEPTION MANAGER

```

E: Exception ex(act: Activity)
A: IF act.Handling_mode IN {Manual, Cooperative} THEN
    Human_Intervention(ex, act)
ELSE
    act.Notify /* notify humans if necessary */
    Resolve(ex, act)
END

PROCEDURE Resolve(ex: Exception; act: Activity)

/* 1. excute all mandatory handler */
FOREACH par IN act.All_parents
    FOREACH r IN par.Binded_Mandatory_ECA_Rules DO BEGIN
        IF ISA(ex, r.Event) AND r.Condition THEN r.Action
    END
END
IF ex.Check = FALSE THEN EXIT /* exception solved */

/* 2. Find procedural handlers */
succ = act.Select_Successors
IF succ <> {} THEN
    FOREACH handler IN succ DO
        Raise(Activity.Execute(handler))
    END
    /* optimistic - assume handler(s) can solve the exception */
    EXIT
END

/* 3. excute ECA handlers one by one until exception solved */
FOREACH par IN act.All_parents
    FOREACH r IN par.Binded_ECA_Rules DO BEGIN
        IF ISA(ex, r.Event) AND r.Condition THEN r.Action
        IF ex.Check = FALSE THEN EXIT /* exception solved */
    END
END

/* 4. built-in WFMS handlers */
WFMS_Handler(ex, act)
IF ex.Check = FALSE THEN EXIT /* exception solved */

/* 5. Re-execution criteria */
CASE act.Reexecution
    Repeatable: BEGIN /* re-execute */
        Raise(Activity.Execute(act))
        EXIT
    END
    Replacable: BEGIN
        /* go back to previous branch point for alternative route */
        FOREACH succ IN (act.Previous_Branch).Select_Successors
            Raise(Activity.Execute(succ))
        END
        EXIT
    END
    Optional: BEGIN /* carry on next step(s) */
        FOREACH succ IN act.Select_Successors
            Raise(Activity.Execute(succ))
        END
        EXIT
    END
    Critical: BEGIN
        Raise((act.Parent).Abort)
        EXIT
    END
END

/* exception not resolved */
Human_Intervention(ex, act)
END

```