Technical Report, LSDIS Lab, Computer Science, University of Georgia, April 10, 2002.

Exception Handling for Conflict Resolution in Cross-Organizational Workflows

Zongwei Luo[1,2], Amit Sheth[1], Krys Kochut[1], and Budak Arpinar[1]

415 GSRC, LSDIS Lab[1]
The University of Georgia
Athens, GA 30602

IBM T.J. Watson Research Center[2]
PO Box 218
Yorktown heights, NY 10598

Abstract. Workflow management systems (WfMSs) are being increasingly deployed to deliver e-business transactions across organizational boundaries. To ensure a high service quality in such transactions, exception-handling schemes for conflict resolution are needed. The conflicts primarily arise due to failure of a task in workflow execution because of underlying application, or controlling WfMS component failures or insufficient user input. So far, little progress has been reported in addressing conflict resolution in cross-organizational business processes, though its importance has been recognized. In this paper, we identify the exception handling techniques that support conflict resolution in cross-organizational settings. In particular, we propose a novel, "bundled" exception-handling approach, which supports (1) exception knowledge sharing – sharing exception specifications and handling experiences, (2) coordinated exception handling, and (3) intelligent problem solving – using case based reasoning to reuse exception handing experiences. A prototype of this exception handling mechanism is developed and integrated as a part of the METEOR Workflow Management System. An evaluation of our approach is also presented through some sample workflow applications.

Keywords: Cross-organizational business process, responsibility based exception handling, workflow management systems.

1. Introduction

Workflow technology has long been considered as an essential technique to integrate distributed and often heterogeneous applications and information systems as well as to improve the effectiveness and productivity of business processes. With the advent of e-commerce, business processes involving business-to-business and business-to-customer activities usually span across multiple organizations. This requires that Workflow Management Systems (WfMSs) provide a set of tools supporting the necessary services for workflow creation, workflow enactment, and administration and monitoring of these business processes in cross-organizational settings. One promising technique to realize support for cross-organizational processes is process outsourcing, usually through contracting. The coordination of processes across organizational boundaries requires policies such as contract setup, contract fulfillment, and conflict resolution to realize the process outsourcing activities [1].

A cross-organizational processes is realized by connecting processes across organizational boundaries. There are primarily three ways of constructing cross-organizational business processes:

- *Split and deploy*: A whole process is designed, and then split into several sub-processes, and deployed in different organizations.
- *Composition*: A whole process specification is composed from several parts that may be contributed by different organizations. Then the process is built and deployed.
- *Outsourcing*: This approach uses the idea of contracting, where several process parts are outsourced. An entire (possibly incomplete) process specification is usually available beforehand.

To meet the business demands and needs, the idea of contracting is used to capture the semantics in process interactions. The contracts are setup in such a way that the processes should meet the requirements specified in the contracts. More specifically, a contract describes several constraints on the outsourced process including expected outcome of the outsourced process, the quality of service (e.g., execution time, throughput, etc.), and rules indicating conflict or exception types and their handling methodologies.

Service name:	Registration
Service description:	Level 3 network users registration process
Provider responsibility:	Availability and performance
Service charge:	Memebership fee
Target availability:	24*7*365
Expected availability:	24*7*365
Failure response:	10 seconds

Figure 1 A contract for registration process

Here is an example for a contract regarding an outsourced registration process (see Figure 1). This sample contract lays down the following agreed items for users to use the registration process:

- Service provider's responsibility is to ensure the service availability and performance.
- The service is expected to be available 24 hours a day, 7 days a week, and 365 days a year.
- The response to the service failure should be given within 10 seconds.

When these requirements stated in the contract could not be met, it is called an abnormal situation, which might be caused by errors, faults, and incompetence to fulfill service, or business rule changes. Abnormal situations in cross-organizational processes can be classified as follows:

- A contract cannot be fulfilled.
- A contract may be compromised.
- A contract needs to be modified.
- A contract needs to be terminated before it expires.

In case any of these abnormal situations occur, they must be resolved such that the contracting partnership can be maintained and both parties can benefit from the partnership. That is, we need to have a solution of conflict resolution to support service fulfillment.

Key contributions of this research are in the area of conflict resolution by supporting workflow exception handling that supports business processes across organizational boundaries. While earlier work has addressed the issue of specifying interactions in cross-organizational workflows, we provide a comprehensive solution to manage these interactions, especially when conflicts arise during the execution of such processes. In particular, we discuss a novel exception handling mechanism for conflict resolution in cross-organizational processes. It bundles knowledge sharing, coordinated exception handling and intelligent problem solving. A compensation preceding rework (CPR) is used to generate an exception-handling template in handling cross-organizational exceptions. The template contains information with default values for describing the abnormal situation and necessary actions to be taken to handle the abnormal situation. Such a template will be populated during the exception resolution process to contain an exception-handling scheme. The intelligent problem solving components are used to populate this CPR template to adapt to various exceptional situations. For example, the adaptation capabilities are provided by human input as well as structured process interaction analysis. Thus, the human interaction is made possible in the whole exception handling process. And more importantly, several process interaction patterns are identified to improve the adaptation capability for the exception handling processes.

To automate the exception handling process, we have identified five coordination patterns of exception handling processes. These patterns facilitate a flexible exception handling mechanism among different organizations. These patterns while facilitating automatic exception handling processes, provide a more complete picture about the abnormal situation by bring both local and remote exceptions into consideration when dealing with exceptional situations.

In this paper, an exception handling mechanism is presented to help conflict resolution in cross-organization workflows. This mechanism is unique in that it reconciles information from both service providers and requestors to derive a resolution scheme. It combines technologies from several fields such as business process management and artificial intelligence to provide a comprehensive solution for resolve cross-organizational conflicts.

The organization of this paper is as follows. In Section 2, we discuss targeted problems in the paper. In Section 3, related work is reviewed. We then propose our strategy in Section 4. In Section 5 through 7, our exception handling solutions, namely knowledge sharing, coordinated exception handling, and intelligent problem solving are presented. In Section 8, a detailed example is illustrated to demonstrate the approach presented in this paper. In Section 9, we discuss our implementation. In Section 10, our exception handling mechanism is evaluated. Finally a summary and a future work are presented.

2. Targeted Problems

In this section, we discuss the targeted problems in exception handling in cross-organizational business processes using an example from the telecommunication industry. Telecommunication business sector is one of the important fields for studying cross-organizational business processes. In this sector, process-

outsourcing activities have been very common recently. Many telecommunication infrastructure providers often contract out their unused bandwidth to other telecommunication service providers. To facilitate the bandwidth outsourcing, cross-organizational workflow techniques are being considered.

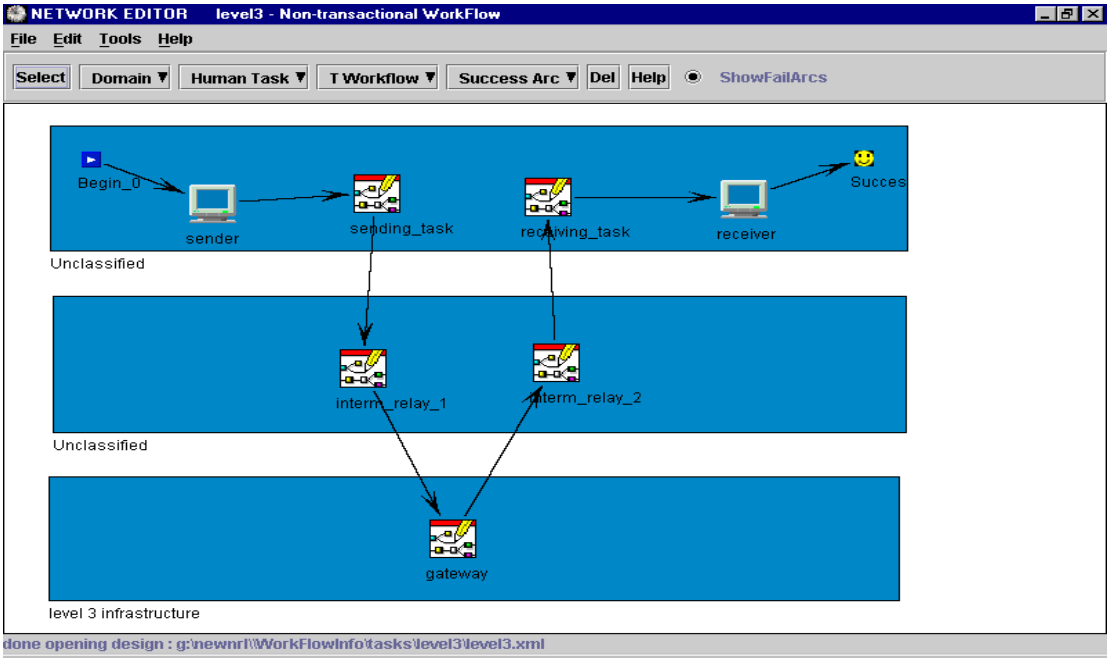

Figure 2 A Cross-organizational workflow: telecommunication bandwidth outsourcing

An example of such cross organization workflows is shown in Figure 2. In this example, a telecommunication infrastructure provider, Level 3 (L3) offers on-demand bandwidth service to let contracted service providers (SP), such as ISP and DSL providers, accommodate their customers' new data and voice applications. The interfaces to L3's network infrastructure are called Gateways. These gateways facilities providing co-location space where Web-centric customers can physically locate their equipment and connect directly to L3's network. Gateways also house L3's routers, transmission equipment, and soft-switches to allow interconnection with other local and long distance networks. L3's direct customers are other SPs. These SPs sign contracts with L3 to use L3's infrastructure. L3 also allows these SPs to change their bandwidth according to their customers' needs. L3 can either accept or reject the requested bandwidth change. If the request is rejected, the rejection response must come within one minute. This requirement is specified in the Service Level Agreement (SLA) between L3 and an SP. If the rejection response does not arrive in one minute, a time out exception will be raised at the task of inter_relay_1. Handling this exception requires cooperation between processes in L3 and SP. SP needs to share the exceptions received with the L3 to find out exactly what might be a cause. By linking the SP's exceptional data with L3's exceptional data, the cause could be identified in a more informed manner. Thus during the exception handling process, information exchange is needed. This means in many cases, exceptions in these cross-

organizational settings must be handled in a cooperative manner between participating processes. The cooperation will be accomplished by using exception-handling processes.

Other exceptions in this application could be related to an incorrect input provided by service requestors as well as system level errors. Various problems in handling exceptions across organizational boundaries exist, mainly in three areas:

- *Specification*. Some exceptions defined in one organization (e.g. SP) are not defined in another organization (e.g. L3). For example, a timeout exception type is defined in SP but not defined in L3. This makes it more difficult to link the SP's exceptional situation with L3's exceptional situation. The exception format may be different in different organizations (e.g. L3 and SP). For example, exception types don't match. Some exceptions have different semantics (meaning or interpretation) in different organizations.
- *Propagation*. Some exceptions are local to an organization (e.g. L3), even if they are related to another organization (e.g. SP), which is unaware of its occurrence. For example, SP could send in a registration request with incorrect input. The incorrect input is finally captured in L3 and handled locally in L3. This is not ideal because it makes the service requestor use what is not correct. Thus, this exception captured in L3 should be routed to SP so SP could improve its way of doing business. However, exception propagation path is usually not well defined in cooperating organizations (e.g. L3 and SP). When an exception caused by one organization (e.g. SP) is detected by another organization (e.g. L3), it is possible that enough information may not be available to verify it. This might result in serious problems. For example, L3 handles an incorrect input with a large amount of resources and SP does not want to compensate it for the simple reason SP is not aware or able to verify the exception.
- *Handling*. Different organizations (e.g. L3 and SP) may have different views for some exceptions. Thus the handling policy may be different. This could create problems. For example, for a time out exception, SP handles it by re-submitting the service request. In L3, it will always raise a time out exception to SP if it encounters the same incorrect input pattern from SP. The cycle could go for unlimited times.

We classify these problems of handling exceptions in cross-organizational settings into the following categories:

- *Heterogeneity*. This is related to different exception specifications, propagation schemes, and handling policies for different organizations as mentioned earlier.
- *Responsibility determination (coordination)*. Local exception handling is not always the best solution for handling exceptions in cross-organizational settings. For example, in some cases, exceptions need to be routed from L3 to SP or vice versa to their responsible parties. Such exception routings across organizational boundaries are currently missing.
- *Black-box effect*. This refers to lack of understanding of the outsourced processes. This is because those outsourced processes usually are considered as black boxes, where their internal details are

encapsulated. For example, execution status of an outsourced process may not be available. In this case, if the service requestor, e.g. SP wants to terminate the process running in L3, it is hard to do so without knowing the process execution status.

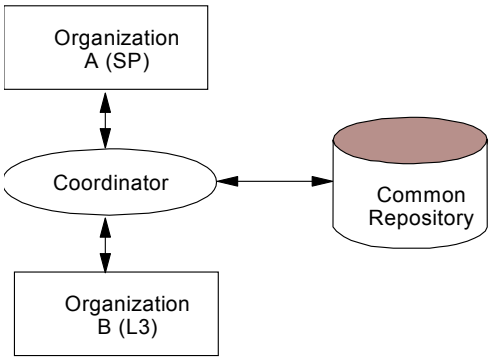

Figure 3 Conceptual exception handling scheme

The conceptual exception handling architecture is shown in Figure 3. Exceptions could occur both in SP and L3. Consider a simple exception to show the exception handling scheme. For example, when L3 receives an incorrect service request from SP, it will not try to handle it. Instead, L3 will also try to route the exception to the SP. The common understanding about the exception caused by the incorrect input will be captured in a common repository. The coordination happens when L3 captures the error and tries to find a routing to the SP. This is actually responsibility determination. The actual routing will be decided according to the process interaction pattern between the SP and L3. If this exception happens before and a case handling it is captured in the repository, the coordinator will retrieve the handling scheme and route the scheme to corresponding party to execute it. In this case, the SP receives the exception and exception-handling scheme. It will correct the input and re-submits the request to L3.

3. Related Work

In this section, we discuss four areas of related work: workflow exception modeling and specification, exception handling and adaptive workflows, knowledge based exception handling, cross-organizational exception handling.

Some researchers (e.g., [2], [3], and [4]) have identified the importance of incorporating the exception handling mechanism into WfMSs. The role of exceptions in office information systems was discussed at length in [5]. The author presented a theoretical foundation, based on a Petri-Nets, for dealing with different types of exceptions. This work is purely driven by organizational semantics rather than by a workflow process model. In [6], different approaches were presented on workflow exception handling including Event Condition Action (ECA) rules to model expected exceptions [7]. In the same work, a general discussion about exceptions in workflow systems based on objected-oriented databases is also

provided. In [6], taxonomy for exceptions in workflow systems was reported. In contrary to these previous approaches, our intelligent exception-handling system helps to measure the similarities among cases during case retrieval and analysis, where a case in our system is used to document exception handling knowledge.

In [8] an exception handling mechanism employing a combination of programming language concepts and transaction processing techniques was proposed. However, aborting or canceling a workflow task is not always appropriate or necessary in a workflow environment. Unlike a database transaction, tasks in workflow systems could encapsulate diverse operations. The error handling semantics of traditional transactional processing systems are too rigid for exception handling in workflow systems.

3.1 Exception Handling and Adaptive Workflows

The work reported in [9] explored how techniques from intelligent reactive control could be leveraged to provide adaptable capabilities within workflow technologies. Main artificial intelligence technologies used are agent planning and procedural reasoning.

In [10], authors differentiated two classes of exceptions, known and unknown. To improve the business processes, they identified four perspectives: incompleteness, informal aspects, requirements for the distribution of work, and the occurrence of incidents. These four perspectives were interwoven, overlapping, and incomplete. They were derived by means of one's personal point of view. These four perspectives were to be obtained through surveys. Exceptions were defined after these perspectives were obtained. Process designer could further improve the process by changing it, e.g., from structured process to semi-structured process to cope with the exceptions identified.

In [3], an adaptive exception handling scheme was used. In addition to other exception handler candidates, such as retry, recovery, compensation, etc., workflow evolution and modification were also considered as exception handler candidates.

3.2 Knowledge Based Exception Handling

In [10], authors offered a flexible exception handling mechanism and moved in the direction to take a knowledge-based approach in dealing with exceptions. They discussed that case based reasoning systems could provide an appropriate support in the usage of a knowledge base. However, in [10], the case base was used for an offline purpose. That is, case base was not used for actively participating in handling exceptions at runtime. It was used only for gathering knowledge. Furthermore, their discussion was not conducted in the context of cross-organizational business processes.

In the work of [11], authors took an active approach in applying case-based reasoning (CBR) in exceptional problem solving. They used a context-dependent approach to support adaptive exception handling. In addition to solving problems in ad-hoc workflows, a CBR-based exception handling system was proposed to collect exception-handling cases, derive exception-handling patterns from the experiences captured in exception handling, and reuse the prior gained exception handling experiences in the future.

The work reported in [12] was more related to coordination science, such as high-level conflict management. This work was based on the Process Handbook project at MIT Center for Coordination Science. It involved development and evaluation of systems for handling multi-agent conflicts in collaborative design [12]. The exception handling discussion in this work was at the level of specification and did not involve execution or run-time issues. A knowledge base where knowledge was acquired totally by human was used in this work to store captured generic process templates. This knowledge base was used only for helping humans to learn to resolve conflicts. This work would be more valuable if generic exception handling expertise in their repository was available and if it can be used directly in a process management system.

In [13], a semi-formal web-accessible repository of exception handling expertise was constructed for the learning purposes. They identified system requirements for exception repository. Their exception taxonomy was comparable to the exception categorization discussed in [3, 11]. This work, however, lacks consideration for system operations and process execution.

3.3 Cross-organizational Exception Handling

In [14] and [15], there are several proposals for inter-connections among WfMSs across organizational boundaries. Until now, there has been little progress in cross-organizational exception handling, except in the CrossFlow project. In CrossFlow, an exception-handling mechanism is used to terminate business processes currently running in another organization [16]. In our approach, we consider this organization has exposed a process termination interface, which can be accessed by the exception handling components [17].

4. Exception Handling Strategy

In this section, we will discuss our strategy for handling exceptions in cross-organizational settings. We will begin with a discussion of intra-workflow exception handling used in the METEOR WfMS.

A try-catch technique is used as an exception-handling mechanism in many programming languages, including C++ and Java. This technique can be used to easily specify handling of runtime errors. It provides a structural programming means for programmers to write code to handle errors. However, this structured try-catch mechanism also puts limitations on exception propagation. That is, an exception can only be propagated along the program's calling sequence. One shortcoming of this approach is that when there is a better handler for an exception, it is very hard for a programmer to write code to propagate it to this handler.

When this try-catch mechanism is used in distributed systems, the focus usually is on local exception handling which still fits the paradigm of structural programming. Usually the exceptions in distributed systems are classified into local exceptions and remote exceptions. Either local or remote exception alone

just provides single side information for the abnormal situation. A combination view of local and remote exceptions will provide much more information to help the exception handling. In this paper, an approach of process-oriented view on exception handling is taken to handle exceptions by gathering information about both local and remote exceptions.

4.1 Intra-WfMS Exception Handling

A WfMS is a distributed systems and it has its own exception handling requirements. In this section, we will focus on the exception handling in METEOR WfMS [18]. Our discussion will cover various aspects of handling exceptions within a WfMS.

State of the activities within a process at a certain point of time is called the process state and is specified through task and data states, as well as the status of the workflow environment. Inter-state dependency constraints are enforced through process transitions. A process can be thought of as a series of process states linked by process transitions with specified starting and ending state.

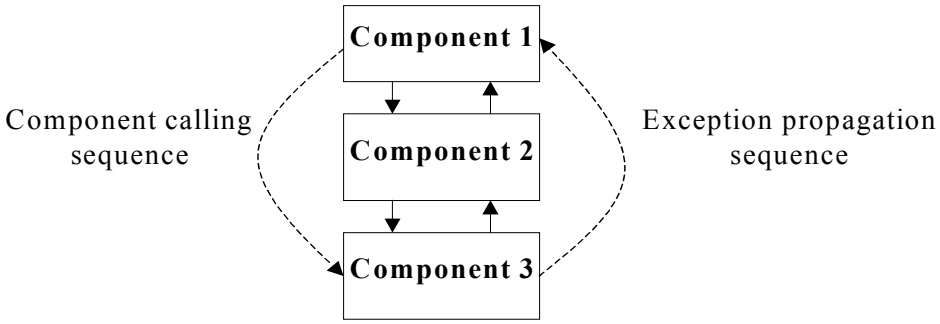

Figure 4 Exception propagation according to calling sequence

In METEOR, an exception is viewed as an occurrence of some abnormal event that the underlying workflow management system can detect and react to by invoking a system or user defined handler. However, an event which can be considered as abnormal, but which is not detected by the enactment workflow management system will not be represented as an exception in METEOR. This is a consequence of the workflow management system implementation. Nevertheless, from the modeling point of view, this abnormal event is still an exception. For example, consider a workflow instance that must be terminated due to some unforeseen event (for example, a customer's company went out of business). In such cases, the workflow management system and/or the workflow application can be notified externally of such an event by receiving an exception signal. In METEOR, such an abnormal event is considered as external exception signal. An exception signal may be sent to METEOR by an external program (for example, a database trigger), or manually, by a workflow administrator.

In METEOR, we have adopted a competence-driven exception handling (see Figure 4). A component that has a handler specified for a given exception is said to be competent to handle the exception. In case

an exception occurs, it is first made available to the workflow component in which the flow of control currently resides. If the component is competent to handle the exception, that is it has at least one handler specified for the given exception type, it handles the exception. On the other hand, a component may also decide to send the exception to a higher level component in the component hierarchy. If a component has not been designed for handling of the exception, the exception is passed onto the higher level automatically.

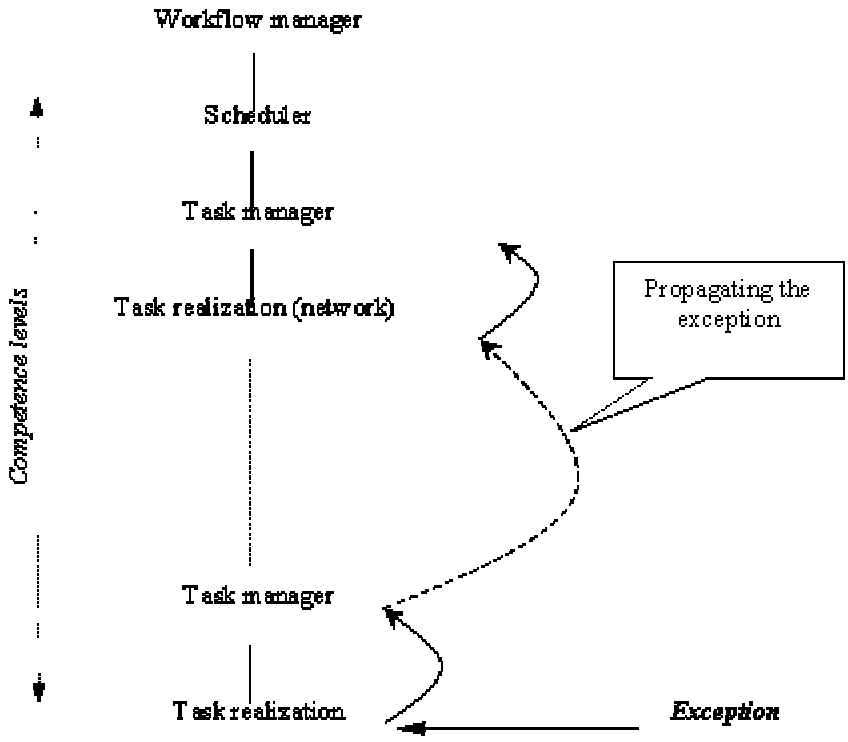

Figure 5. Competence hierarchy

In METEOR, workflow tasks are scheduled by task schedulers, while task executions are controlled by task managers. A task realization performs the actual work of the task (for example a database transaction, or a legacy application). Also, in the METEOR model a workflow task may be comprised of a whole sub-workflow, with its own network of tasks. In fact, the whole workflow application is regarded as a single sub-workflow, controlled by a network task scheduler. All of the mentioned components participating in a METEOR workflow application form a hierarchy, as illustrated in Figure 5.

The METEOR runtime system attempts to find the closest (most competent) component in the hierarchy to handle the exception. The search progresses up in the hierarchy until a suitable component is located. The topmost component, i.e. the workflow manager is designed to handle all of the detected and unhandled exceptions. Of course, this handler is the most general and therefore its handling action is also of general nature.

4.2 Exception Handlers

Various exception handlers in workflow systems are:

- *Ignore*. An exception is ignored, if no actions are taken to handle it.
- *Record*. An exception will be recorded only when it occurs. The storage place can be a log file or a database.
- *Retry*. Only repeatable action can be retried. An example is a database connection. The retry times and the duration of waiting period need to be specified when retry is used.
- *Compensation*. An action can be compensated if another action can compensate its effects.
- *Alternative task*. Instead of executing current task, an alternative task can be executed.
- *Backward recovery*. Backward recovery is used to rollback the workflow to a former consistent state.
- *Forward recovery*. Forward recovery is proposed for workflow recovery in long-running processes. Workflow is rolled back to a certain point, and then restarted to follow an alternative execution path. This point may not necessarily satisfy the global correctness criteria.
- *Propagation*. If no local handlers are available then exception is routed to another workflow component, e.g. an exception handling coordinator.
- *Termination, suspension, and resumption of processes*. They provide the basic functionality for supporting workflow exception handling.
- *Procedural exception handler*. It involves a series of steps to handle exceptions.

The above handlers if used alone are not good candidates for cross-organizational settings. Handlers of Ignore and Record are trivial solutions. Retry usually works in a working system with poor performance, or in special situations. For example, a request to a database server that is restarting may be retried. In most cases, a simple retry will not solve the problems encountered. Compensation and alternative task are other possible candidates for this exception. Recovery based solutions work only in a transactional environment. Therefore, termination, suspension, and resumption of workflow processes provide basic functionality for workflow exception handling. Propagation and procedural exception handlers are too generic unless good templates are available.

Compensation preceding rework (CPR) is developed as exception-handling template for handling exceptions. Here, compensation is rationale based. A compensation decision is reached through certain communication efforts when both parties agree. If the effects of an operation can be totally removed, it is called a perfect compensation. If no action can be taken to decrease the effects, it is called no compensation. Otherwise, it is called partial compensation. The actual CPR scheme will be based on the following items [17]: Process access interface, process monitoring interface, and process controlling interface.

4.3 Cross-organizational Exception Handling

Exception handling research in enterprise-wide workflows has achieved numerous results. For example, the exception handling strategy in METEOR is a more flexible approach than the basic try-catch mechanism for handling exceptions inside a WfMS. The process oriented exception handling involves a series of actions including complicated task and even human interactions to handle exceptions. This feature is very important especially in handling inter-WfMS exception across organizational boundaries. Thus, the exception handler is not just a program any more; it should be an exception handling process.

The exception handling strategy in METEOR lays down the foundation for our research in the area of exception handling across organization boundaries. That is, this strategy makes it possible for us to propose an exception handling strategy on top of WfMS by utilizing the internal exception masking and propagation [3] inside a single WfMS. At each control layer in METEOR model, an exception could be propagated to an entity called exception-handling coordinator. Then this coordinator will handle this exception. Since local handling is not always the best solution for handling exceptions, exception handling across organizational boundaries is focused more on flexible exception propagation once an exception is to be propagated among WfMSs. To propagate exception among these processes deployed in different organizations, the following items must be addressed:

- Process status that is supplied by process monitoring or through inquiry.
- Process context information that provides additional information about the abnormal situations.
- Process controllability that determines the exception-handling scheme.

To support exception handling in cross-organizational settings, we have proposed a bundled solution (see Figure 6) to solve the problems of heterogeneity, responsibility determination (coordination), and black-box effect. This solution involves the following components:

- *Exception knowledge sharing*. It includes exception specification sharing and exception handling experience sharing. The aim is to take initial steps towards the development of a methodology to share multiple and often heterogeneous exceptions and exception handling experiences in cross-organizational settings. The exception knowledge is stored in a case repository (see Figure 6).
- *Coordinated exception handling*. It is the process of resolving exceptions in a coordinated manner. It enables problem solvers from multiple organizations to participate in the exception handling process. The exception handling coordinator (see Figure 6) coordinates the exception handling process. Five patterns of coordinating exception handling in cross-organizational settings are identified. These patterns are immediate, deferred, de-coupled, free, and close. They are based on the identified types of process interactions in cross-organizational settings. These patterns are used to facilitate the construction of flexible exception handling processes across organizational boundaries.

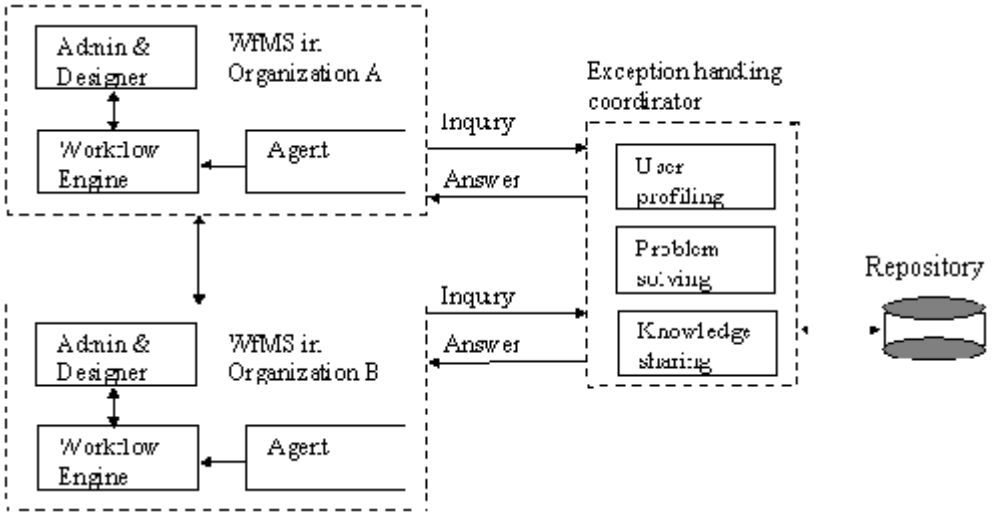

Figure 6 Conceptual model of the coordinated exception handling in cross-organizational settings

- *Intelligent problem solving*. A knowledge-based approach can help workflow designers and participants better manage the complex exceptions that occur during the enactment of a workflow by capturing and managing the knowledge about what types of exceptions can occur in the workflow, how these exceptions can be detected, and how they can be resolved.

Finally, to support exception handling for cross-organizational business processes, we identify the following two requirements to help settle conflicts:

- It is necessary to bring processes back to an equivalent state in case exceptions occur, but not necessarily the same state. The rollback behavior depends on process controlling attributes exposed by the contracting process participants.
- Compensation of previous actions is a good exception-handling candidate. Dynamic generation of compensating schemes based on operation status is always desirable. Compensation schemes may be different for different end users, and different contracting participants.

The Compensation Preceding Rework (CPR) scheme for handling exceptions is proposed partially upon these two requirements. We will discuss more on CPR in later sections.

5. Exception Knowledge Sharing

The exception handling process is an integrated activity that involves both human and automated processes. Knowledge management tools assist human beings involved in the exception handling process. The task of knowledge acquisition and problem solving reflects the theme of continuous process improvement. Exception knowledge is a valuable asset to an organization. It has long been recognized as a major factor determining business competitiveness [19, 20]. It includes the following:

- Exception pattern, which will be identified in the exception specification,
- Exception handler pattern, which will be identified in the exception handler specification, and

- Exception handling experience, which will be acquired in case representations.

These patterns, described using a set of attributes, are stored in the case repository. All cooperating business processes share them. The case repository is built upon the workflow repository. An initial report about this repository can be found in [21].

5.1 Exception specification

Exceptions can be divided according to organizational boundaries. One type of exceptions is enterprise-wide exceptions. The other type is cross-organizational exceptions. Three broad exception categories can be made for these enterprise-wide exceptions: infrastructure exception, workflow exception, and application exception. The infrastructure exceptions and application exceptions are mapped to workflow exceptions that include system exceptions and user exceptions. That is, an infrastructure or application exception will trigger a mapped workflow exception. This mapping scheme is adopted due to the heterogeneous nature of exceptions in applications and infrastructure.

- *Infrastructure exceptions*. These exceptions result from the malfunctioning of the underlying infrastructure that supports the WfMSs. These exceptions include hardware errors such as computer system crashes, errors resulting from network partitioning problems, errors resulting from interaction with the Web, and errors returned due to failures within the Object Request Broker (ORB) environment. In the telecommunication application, an infrastructure exception can be caused by an error in the telecommunication media such as channel.
- *Workflow exceptions*. Two basic sub-groups of workflow exceptions include system exceptions and user-defined exceptions. A variety of system exceptions identify a number of possible system-related deviations in the services provided by the workflow system. Examples of this include a crash of the workflow enactment component that could lead to errors in enforcing inter-task dependencies, or errors in recovering failed workflow component after a crash. User-defined exceptions are specified by the workflow designer and identify possible application-independent deviations in task realizations.

- *Application exceptions*. These exceptions are closely tied to each of the tasks, or groups of tasks within the workflow. Due to its dependency on application level semantics, these exceptions are also termed as logical exceptions. For example, one such exception could involve database login errors that might be returned to a workflow task that tries to execute a transaction without having permission to do so at a particular DBMS. In the telecommunication example, an application exception can be caused by an error in the bandwidth change request that the agents could not be found for the roles required for performing the change assignment.

In the cross-organizational settings, an additional exception category cross-organizational exception should be defined. A cross-organizational exception can be an infrastructure exception, a workflow exception, or an application exception. It can affect the outsourcing fulfillment, and it may not be handled alone in one

outsourcing partner. An example of such an exception in the telecommunication application is due to an error in the bandwidth change request. The agents can't be located for performing the bandwidth change. Because no agents are available to perform the requested services, customers of the SPs are denied services, which directly harms the SPs' business.

5.2 Exception Handling Pattern

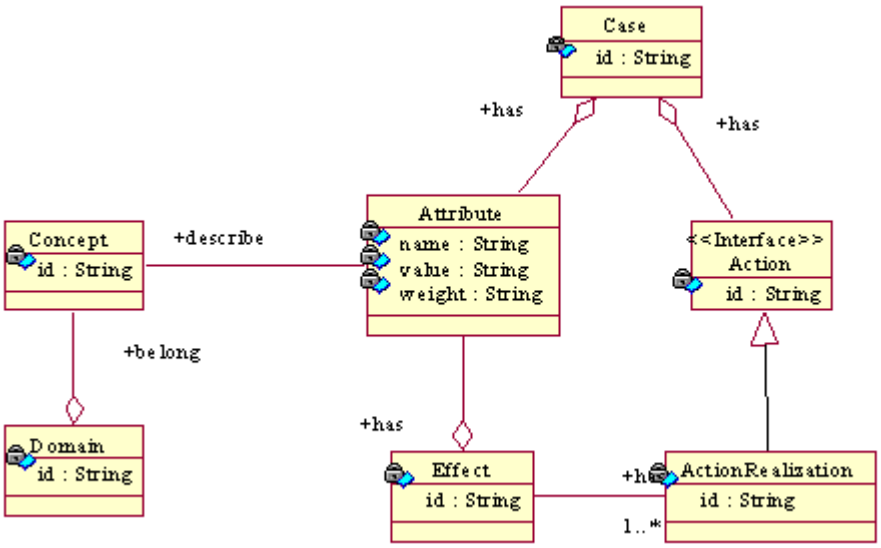

Figure 7 A conceptual model for case

In our approach, exception handling experience will be captured using a data structure called case (see Figure 7). The patterns of the handling process will be derived by using a case-based reasoning (CBR) scheme, or through expert survey. A case consists of descriptions of exception handling knowledge. Each case has a list of attributes specified in the format of <name, value [, weight]>. Name holds the type of the attribute, and an attribute has a value. Weight signifies the importance of the attribute in a case. An attribute can be either mandatory or non-mandatory. Mandatory attributes are essential attributes to characterize the class of a case. Non-mandatory attributes, which are denoted in "[ ]" pairs, provide additional description about the case.

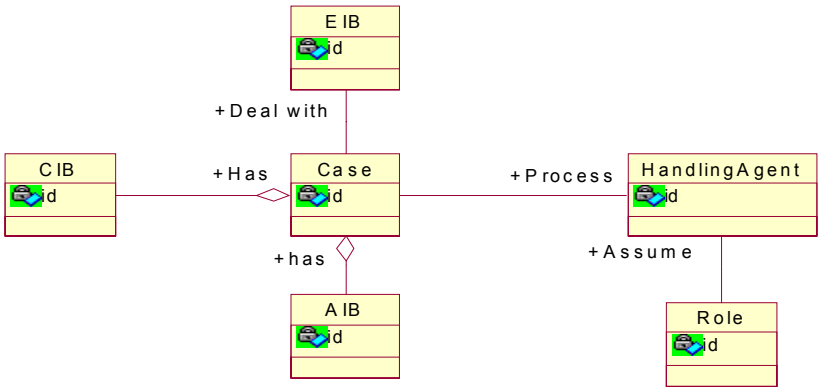

Figure 8 Relationship between case, exception information block and handling agent

A case (see Figure 8) contains three information blocks, namely EIB, CIB, and AIB. An exception information block (EIB) describes an exception situation. A context information block (CIB) records context information about the exceptional situation. It is usually used to capture workflow application data. An action information block (AIB) is used to record the actions taken to handle the exception situation. It contains the CPR exception handling template. It has two blocks: compensation block and rework block. These two blocks are used to describe the exception handling schemes. An example of such a case from L3 application (see Figure 2) is as presented in Figure 9. It describes the experience of handling an exception in which the reference of a task manager couldn't be obtained. The information contained in the CPR template (AIB) actually says to insert the task again into the implementation repository and retry the task.

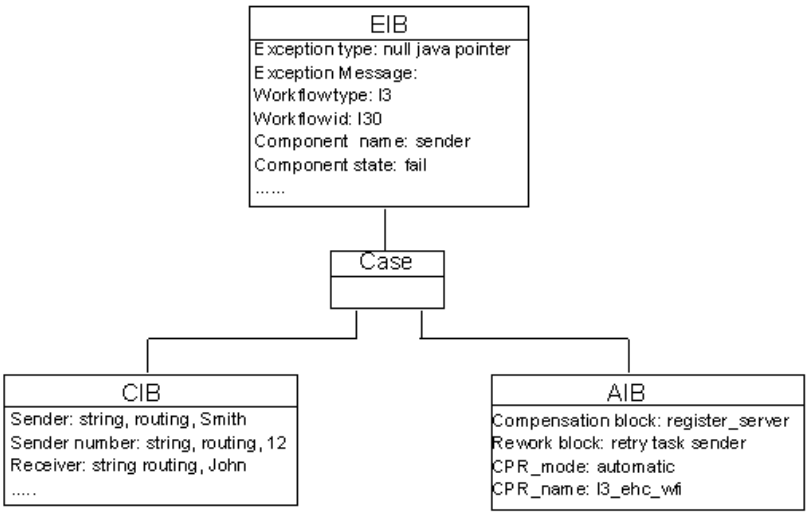

Figure 9 A case for an exception caused by de-registration of a task

The following are the entries in the CPR template:

- Compensation mode. It can be automatic, manual or none.

- Compensation type. It denotes the compensation scheme type. It actually contains the class name of that compensation scheme.
- Compensation action. It denotes the compensation scheme. It actually contains a method name in the class denoted by compensation type.
- Compensation parameter number. It denotes the number of the parameters the compensation scheme needs.
- Compensation parameter string. It holds the parameters to be passed to this compensation scheme.
- Rework mode. It can be automatic, manual or none.
- Rework type. It denotes the type of rework, such as retry, alternative task, etc.
- Rework task name. It holds the task name to be tried or activated.
- Rework state string. It holds the state string so correct task data and parameters can be found. Experts usually don't need to fill in this entry. It would be automatically filled. However, if a task's task data or parameters are unknown, this should be filled by workflow designer. For example, if the workflow designer inserts a new task, and wants to execute this task instead of retrying an existing task, the designer needs to supply the state string. It is usually in the form of "start@start".
- Rework parameter string. It holds the task parameter string so correct task data and parameters can be found. Experts usually don't need to fill in this entry. It would be automatically filled. However, if a task's task data or parameters are unknown, this should be filled by the workflow designer. For example, if the workflow designer inserts a new task, and wants to execute this task instead of retrying an existing task, the designer needs to supply the parameter string. It is usually in the form of "r@start@start".
- Rework host. It denotes which host the task will reside. This information is needed when a new task is inserted and is the executing target.
- CPR mode. It has two modes, "user" and "automatic". If it is in "user" mode, the values filled by experts in the rework state string and parameter string will be used first before the values automatically filled are used. If it is in "automatic" mode, the values filled by experts in the rework state string and parameter string will be used only if the task data and parameters can not be obtained by using the string and parameter values that are filled automatically.
- CPR name. It denotes the adaptability of this CPR. Usually there are four levels of adaptability denoted by strings of "_ehc_wfa", "_ehc_wfo", "_ehc_wfi" and "ehc_wft". If the CPR name ends with "_ehc_wft", it denotes the CPR scheme is adaptable. This CPR scheme can be used without any human intervention and can be applied across tasks, instance, and workflow types. For CPR names end with other than "_ehc_wft", the situation is more complicated. A CPR with a name ending with "_ehc_wfa" is not adaptable at all. A CPR with a name ending with "_ehc_wfi" is adaptable only for this same task in the same workflow type. A CPR with a name ending with "_ehc_wfo" is limited in adaptation. Human involvement is needed to be present to make changes to the CPR.

The control flow semantics of the CPR is as follows:

- If (compensation mode is automatic) the compensation scheme is automatically executed.
- If (compensation mode is manual) a human needs to be involved to execute the compensation scheme.
- If (compensation mode is none) no compensation is necessary for this CPR.
- If (rework mode is automatic) the rework scheme is automatically executed.
- If (rework is manual) a human needs to be involved to execute the rework scheme.
- If (rework is none) no rework is necessary for this CPR.

6. Coordinated Exception Handling

A competency-based mechanism works in the single organizational environment by propagating exceptions among runtime system components such as task manager, task scheduler, and workflow manager. To support exception detection and propagation, the exception handling coordinator will record the cross-organizational exceptions, and share this information with co-operating business processes. An exception handling mechanism that is able to route exceptions across organizational boundaries is needed. To realize this kind of exception routing, WfMSs need to report cross-organizational exceptions to the exception handling coordinator. The coordinator, then, will share this information with co-operating business processes.

To illustrate the requirement of routing exceptions across organizational boundaries, we provide the following example. In the L3 telecommunication application (see Figure 2), service providers act like sales departments to Level 3. Level 3 acts like production department (service fulfillment) to service providers. Bound by the principle of "customer satisfaction is the first priority", sales personnel sometimes will receive special orders from customers. Sales department then needs to forward these orders to the production department to allocate more bandwidth if its subscribed bandwidth cannot meet its customers' demand. The production department may need additional manpower and resources to fulfill the orders. Due to additional cost incurred for those special orders, the performance level of the production department will be lower than expected. Who will be responsible for such poor performance? We name our solution to this problem responsibility based exception handling. The proposed technique works as follows:

- Sales department can forward any orders to production department. Production department will fulfill the orders.
- Additional cost (considered as an exception) other than normal will be charged to sales department instead of production department.

In this way, the sales department is encouraged to improve their capability in identifying profitable orders. This approach makes a healthy business relationship possible between L3 and SPs.

In the following, we will illustrate five modes of coordinated exception handling for constructed flexible exception handling processes across organizational boundaries. These five exception-handling patterns are based upon possible process interactions. Generally there are two basic types of process

interaction: *one-way interaction* and *two-way interaction*. These two basic types of interactions can used to represent more complicated interactions. In the one-way interaction, one process can send a message (request) to another process. After that, no interaction exists between the two processes. In the two-way interaction, one process can send a message (request) to another process. It can either receive response immediately or at a later time.

6.1 Immediate Pattern

Immediate cross-organizational exception handling pattern is appropriate when there is a single two-way interaction point between processes deployed in different organizations. The exception information is exchanged through a single interaction point. For example, as shown in the Figure 10, SPs route their bandwidth change requests to L3 due to their customers need more bandwidth. They will immediately get an exception in case L3 determines that not enough funds are supplied along with the request. In this pattern, service requestor (SP) is waiting to get the response from the service provider (L3).

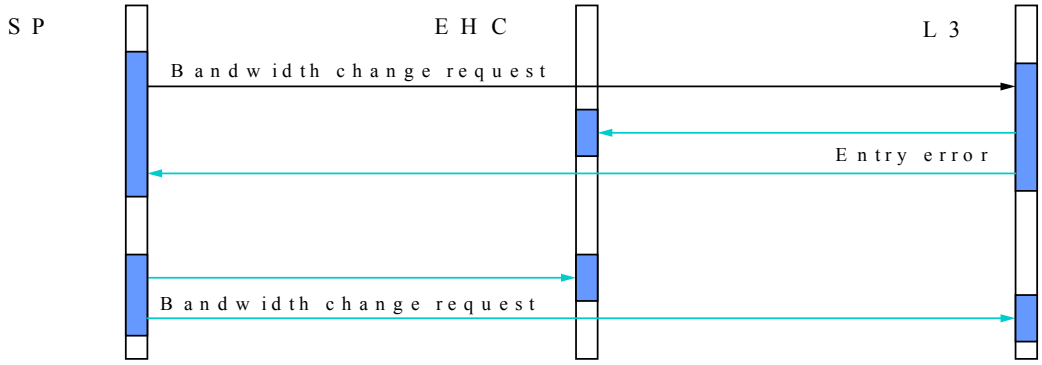

Figure 10 Immediate pattern of cross-organizational exception handling

6.2 Deferred Pattern

Deferred cross-organizational exception handling pattern is used when there is a single two-way interaction point between processes deployed in different organizations. The difference from the immediate handling pattern is that in the deferred mode an exception is not reported immediately to the cooperating process when it occurs. For example, as shown in the Figure 11, Customers need to subscribe to use SP's services. SP will not raise exception immediately to customers when it determines that customers' credit history is not verifiable during the request. Instead, SP will fulfill the request, and later raise the exception to customers through the same two-way interaction point. Same as in the immediate pattern, service requestor (Customer) is waiting to get the response.

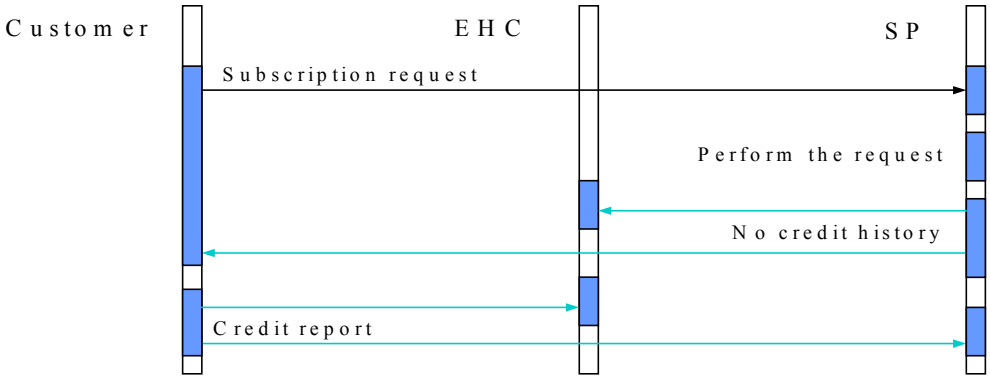

Figure 11 Deferred pattern of cross-organizational exception handling

6.3 De-coupled Pattern

De-couple cross-organizational exception handling pattern is appropriate when there is only a single one way interaction point between processes deployed in different organizations. For example, as shown in the Figure 12, SPs route their bandwidth requests to L3 through a one-way interaction point. L3 will try to fulfill the request. However, when L3 finds that the requested 513 Kbps channel is not available, it needs to raise this exception to SPs. Since there are no other interaction points between them, L3 needs to raise it to SPs through the exception handling coordinator. In this pattern, service requestor (SP) does not need to wait to get the response from service provider (L3).

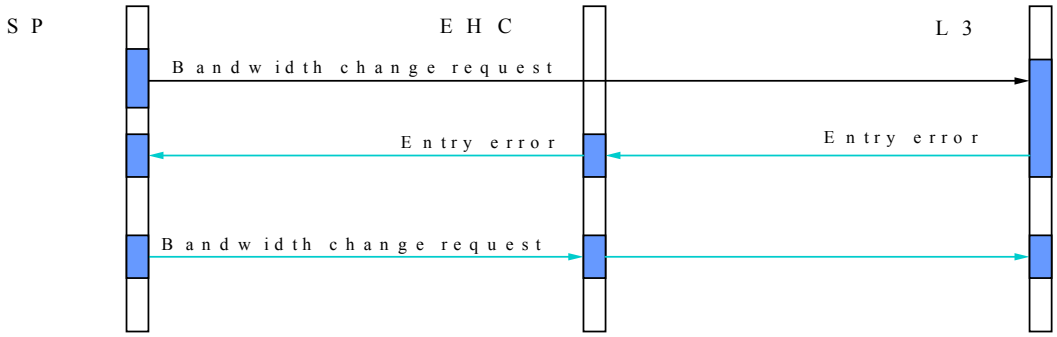

Figure 12 De-coupled pattern of cross-organizational exception handling

6.4 Free Pattern

Free exception handling pattern is appropriate when there are several interaction points between processes deployed in different organizations such that two-way interaction is possible through more than one interaction point. For example, in the Figure 13, SPs route its customers' bandwidth requests to L3. L3 will not raise exception to SPs if it determines that not enough money is supplied along with the request. Instead, L3 will fulfill the request, and later raise the exception to SPs through another interaction point determined by the exception handling coordinator. Same as in decoupled pattern, service requestor (SP) does not need to wait for the response from service provider (L3).

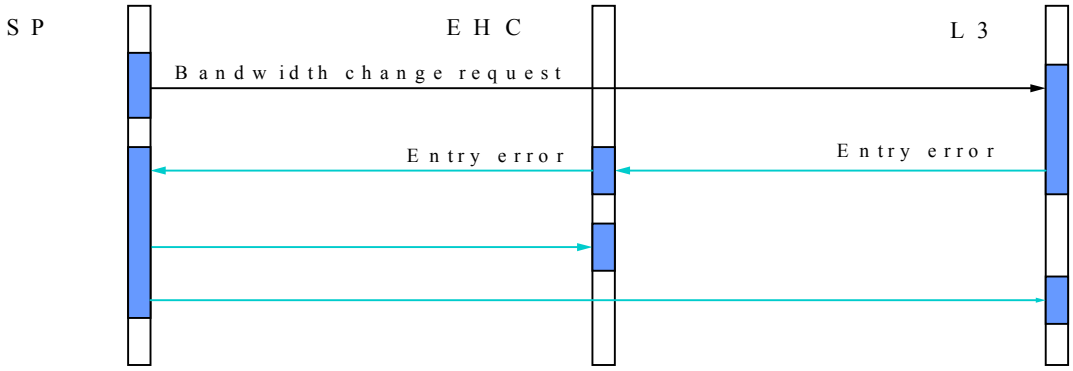

Figure 13 Free pattern of cross-organizational exception handling

6.5 Close Pattern

Close pattern cross-organizational exception handling is used when there are no interaction points between processes deployed in different organizations such that no interactions are possible. For example, in Figure 14, SPs determines the service quality provided by L3 is not satisfactory, and if interaction between them is not possible at this time, SP will raise an exception through exception handling coordinator (EHC). EHC can at least record the exception for later use.

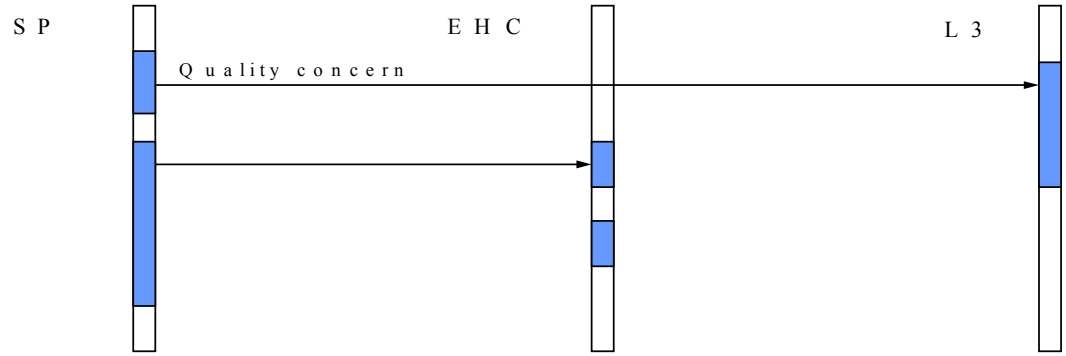

Figure 14 Close pattern of cross-organizational exception handling

7. Intelligent Problem Solving

Our exception handling system uses a case-based reasoning (CBR) [22] based approach for managing the exception handling knowledge. This CBR mechanism is used to improve the exception handling capabilities. We have taken an approach of concept based case management [3]. When an exception occurs and is propagated to the intelligent problem solver, the case repository will be consulted. Similar cases will be retrieved and analyzed. Solutions to the exception will be automatically derived if a similar case is adaptable. For CBR to work, similarity among cases must be carefully determined. In our approach, the retrieval procedure of similar previous cases is based on the similarity measure that takes into account both semantic and structural similarities between the cases. A similarity measure is achieved by obtaining the following:

- *Exception similarity*. Exception similarity is based on the is-a relationship in the exception hierarchy in METEOR [18].
- *Workflow similarity*. It is the workflow structural similarity such as AND, OR building block similarity.
- *Context similarity*. It is obtained by computing nearest neighborhood function of the quantified degrees of semantic similarities over workflow application data. To do so, domain experts build a concept tree. Then the tree is stored into the case repository.

An example of such a three layer case match is shown in Figure 15 These two exception cases are related to the L3 telecommunication workflow application (see Figure 2). The details of similarity measure is described in the next subsection.

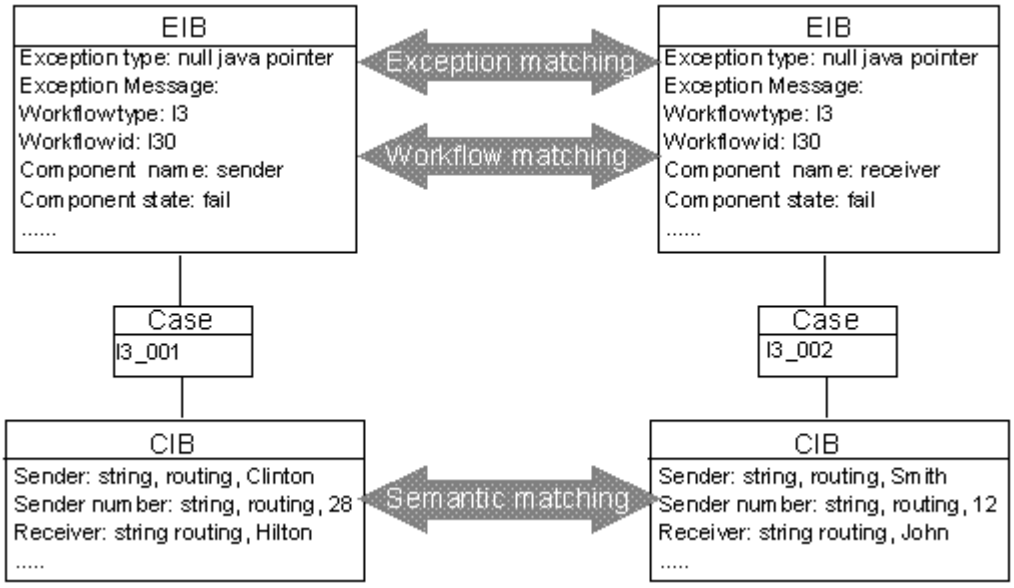

Figure 15 Three layer matching among two cases

In our CBR based approach, information about previous problem solving cases is retrieved to help solve new problems. During the workflow execution, if an exception is propagated to this CBR based exception-handling component, the case-based reasoning process is used to derive an acceptable exception handler. Human involvement is needed when acceptable exception handlers cannot be automatically obtained. Solutions provided by a person will also be incorporated into the case repository. Effects of the exception handler candidates on the workflow system and applications will be evaluated. Thus, when the exception is handled necessary modifications to the workflow systems or applications can be made. The exception resolution process is actually the population process of CPR templates. The actual exception resolution performs the following tasks [17]:

- The coordination mode of exception handling will be determined. The coordination mode will be determined according to the type of process interactions between business processes.

- The contacting party as well as interaction point will be determined. A contacting party is one of the entities that are responsible for handling exception in the processes in its organization. An interaction point is where the interactions can take place.
- The compensation scheme will be found if necessary. The nature of the processes will affect the compensation schemes. Human involvement is allowed in determining the compensation schemes.
- The rework scheme will be found if necessary. Rework scheme is the plan for the processes to make progress from the faulty points. Human involvement is allowed in determining the rework schemes.

7.1 Case Retrieval

The retrieval procedure of similar previous cases is based on the similarity measure that takes into account both semantic and structural similarities and differences between the cases. A similarity measure is achieved by obtaining the following:

- Exception similarity. Exception similarity is based on the is-a relationship in the exception hierarchy in METEOR model 3.
- Workflow similarity. It is the workflow structural similarity such as AND, OR building block similarity.
- Context similarity. It is obtained by computing nearest neighborhood function of the quantified degrees of semantic similarities over workflow application data. To do so, a concept tree should be built first. The distances between concepts will be stored into the case repository.

To conduct a similarity based case retrieval, the similarities should be computed between targeted case (new situation) and old cases for three components: exception information block, case information block, and action information block. Each component may have its attributes that are application dependent. They are weighted according to the similarity measure algorithms used. We are using the least square distance function to calculate the case similarity. The least square distance function is defined in Figure 16. Assuming two cases a and b both of which have n attributes. The similarity (S) between a and b will be calculated as in Figure 16:

$$S = \sqrt{\sum_{n} wi \times (ai - bi) \times (ai - bi)}$$

Figure 16 Similarity equation for two cases

As shown in Figure 16,

- w_i is the weight for i^{th} attribute.

- a_i is i^{th} attribute in case a.
- b_i is the i^{th} attribute in case b.
- $(a_i\text{-}b_i)$ is the similarity between attribute a_i and b_i.

In case two exception cases do not have the same number of attributes, a default-reasoning scheme [Luo et al 2000] is used. That is, default values will be assigned to these attributes that don't existing in one of the cases before similarity is calculated. The quantified value of similarity about each attribute between two cases is based on the concept. Concept trees built upon ontology are used to calculate the concept similarities. The similarity between two concepts is determined by considering the sibling difference (S-similarity) and the level difference (P-similarity) between the two concepts in the concept tree. For example, assuming that the concept tree is built such that:

- The S-similarity between two concept a and b is 1.
- The P- similarity between two concept a and b is 1.

Then the similarity between the two concepts a and b is 2 by adding both S-similarity and P–similarity. These similarities will be computed and stored into case repository. Queries are designed to dynamically load up the similarity values into the memory during the case match. An example of search for the similarity value between two concepts is as follows:

> Select similarity from concept_tree_name where concept_1 = concept_input1 and concept_2 = concept_input2.

Concept_input1 and concept_input2 are the two concepts between which the similarity is searched. Concept_1 and concept_2 are the column name in the concept_tree_name table.

8. Exception Handling System

The exception handling system (see Figure 17) is implemented based on METEOR OrbWork runtime system. The whole system is design as a separate module to the OrbWork.

The actual exception handling system is the exception handling coordinator, which consists of four servers implemented as CORBA objects, exception handling coordinator, case server, database server, and agent. When the exception handling coordinator is used with OrbWork system, other tools are also used to facilitate the exception handling process. They are the system monitor and network designer. The system monitor reports system status. Network designer, while being used to design workflow applications, is used to change processes.

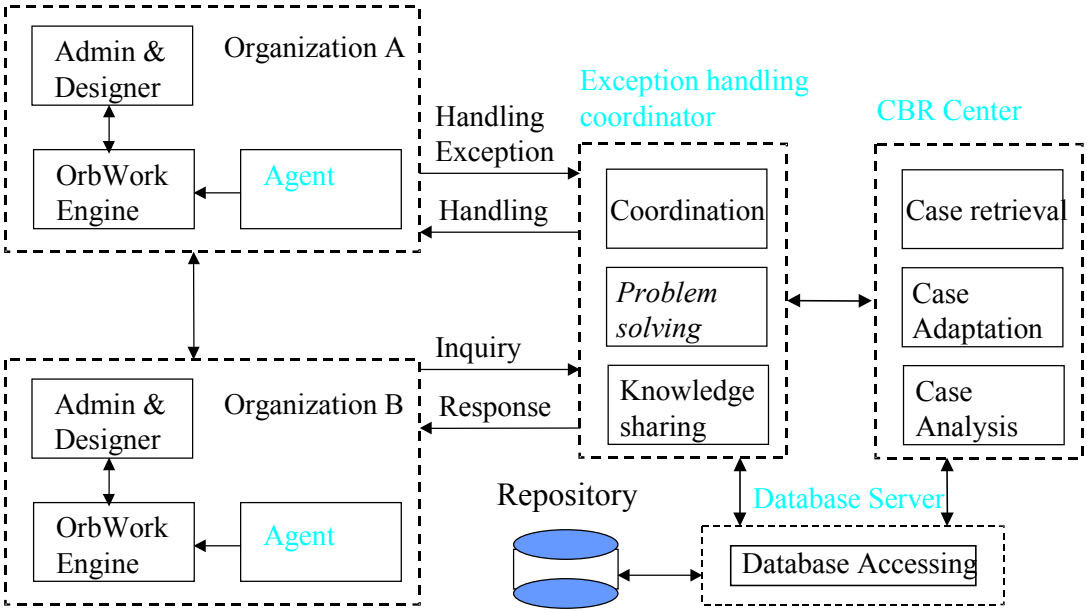

Figure 17 Exception handling implementation conceptual architecture

This coordinated exception handling mechanism has been integrated with METEOR OrbWork workflow management system. It is implemented in a distributed manner (see Figure 18). A set of exception handling protocols (e.g. Handling, HandleException, Inquiry, and Response) has been created (see Figure 18). "HandleExcpetion" is used for WfMSs to propagate exceptions to the exception handling coordinator. "Handling" is used for the exception handling coordinator to send out CPR exception handling schemes. "Inquiry" is used to send a query to exception handling coordinator. "Response" is used by the exception handling coordinator to answer inquires. This exception-handling framework consists of four CORBA servers described in interface description language (IDL). These servers can be distributed over different hosts. They are described as follows:

- CBR Center server. It provides interfaces that allow workflow management systems to propagate exceptions to it, accept exception inquires, and send out CPR exception handling schemes.
- Case server. It provides case operation interfaces such as case retrieval, case adaptation, and case storage.
- Database server. It provides database operation interfaces that encapsulate the database access differences in database systems used in the exception handling system by using JDBC.
- Agent server. It provides interfaces that accept CPR exception handling schemes from Center server and execute the schemes.

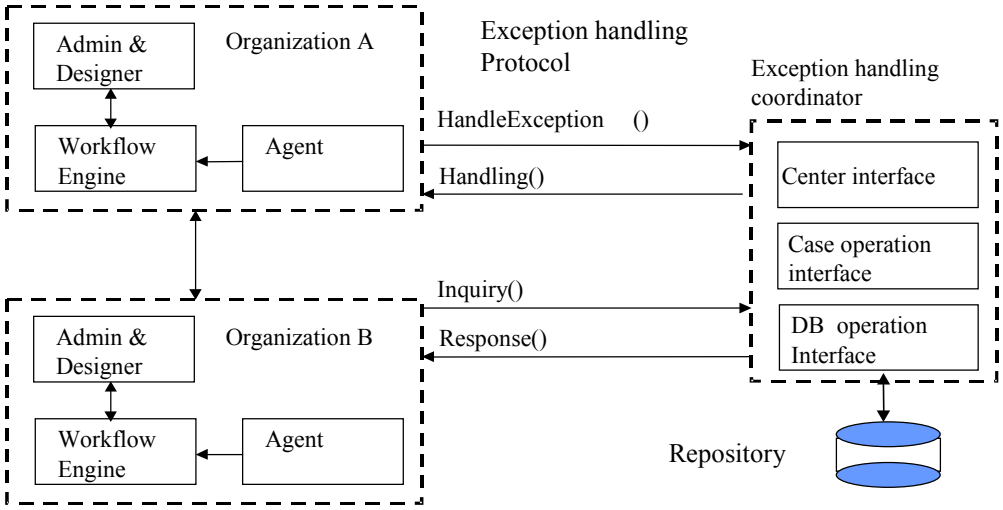

Figure 18 Exception handling coordinator implementation architecture

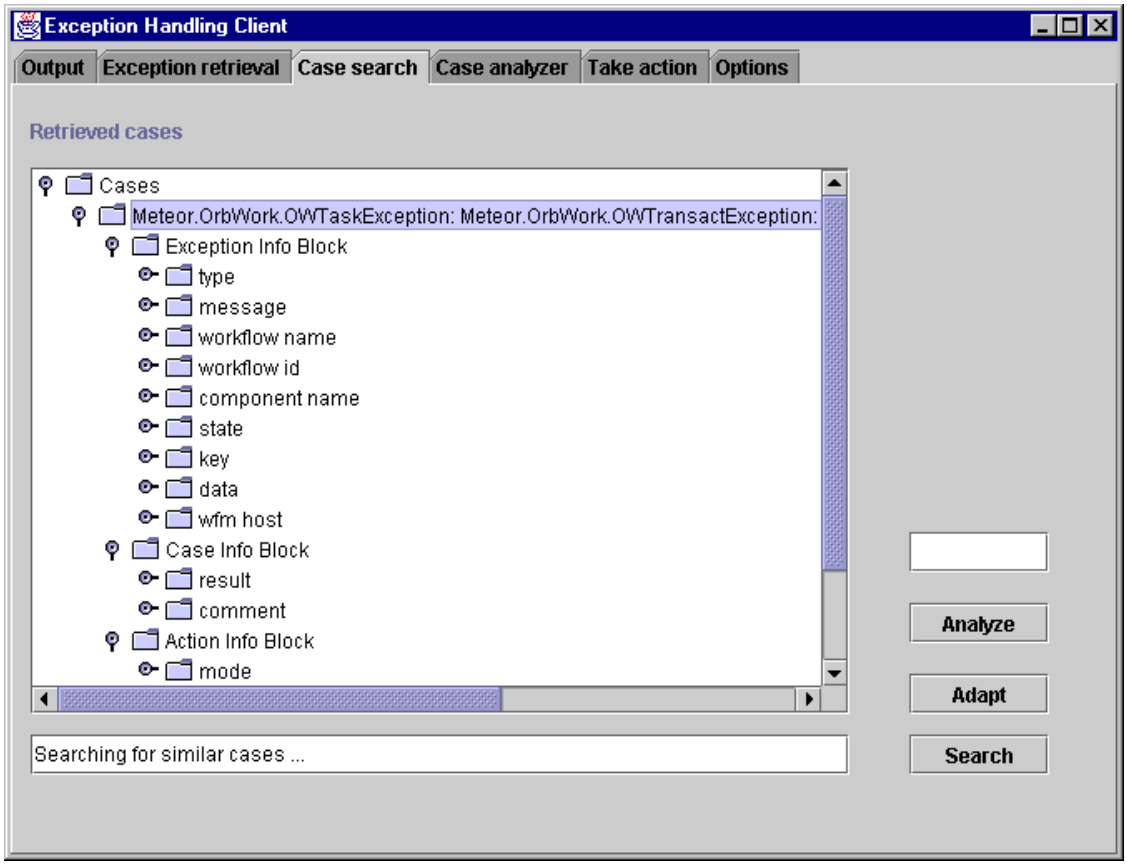

Figure 19 GUI based Exception handling client

A GUI client (see figure 19) has been developed so human beings can interact with the OrbWork runtime in handling exceptions. The GUI client communicates with exception handling coordinators via CORBA IIOP protocols. We summarize how an administrator can use this GUI tool in handling exceptions in the following:

In the exception retrieval panel (see Figure 19), administrators can retrieve exceptions propagated to the exception handling coordinator, which are not handled yet. Administrators can select to handle any retrieved exceptions. Once an exception item is selected, other administrators cannot select it. But other administrators can still view that exception item.

In the case search panel (see Figure 19) administrators can retrieve similar cases. Administrators are allowed to use the interactive search tool to find more case according to his/her interests. When administrators need to modify existing cases, they need to use the case analysis tool available in the analyze panel (see Figure 19).

The case analyzer is used to analyze these retrieved cases. Administrators can insert or remove cases from the case pool. Administrators can also modify the attributes of any cases if allowed. Administrators can interact with CBR based problem solver to solve problems by using the case adaptation tool available in the take action panel. The case adaptation process will begin when the "Evolve" button (which is not shown in Figure 19) is clicked. The suggested result will be displayed when adaptation finishes. Administrators can decide to use the suggested solution by clicking "Take Action" button (which is not shown in Figure 19). Administrators can also write back the cases into the case repository.

8.1 Process instance and data backup

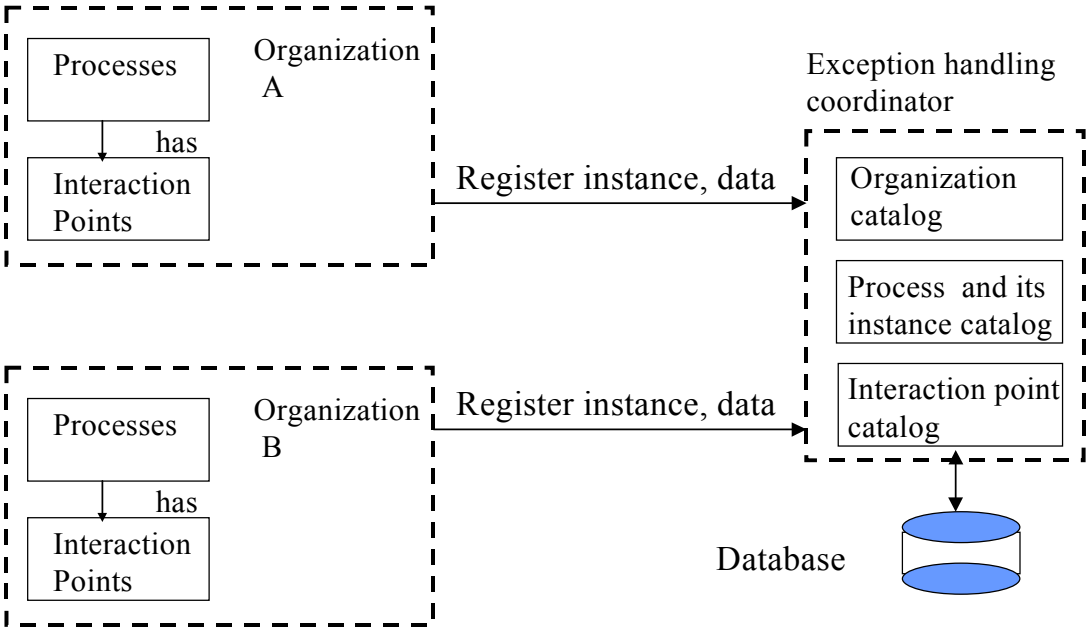

Figure 20 Process instance and data registration

A workflow instance can't be recovered if its instance and data can't be recovered. From time to time, process instances and their data must be stored. A full-scale checkpointing is one of the choices to facilitate recovery process. In this work, an alternative scheme of instances and data backup is taken and implemented. Two reference copies of the process instances and data are maintained. They are used by the

agent to get task data and parameters when the agent receives exception-handling requests from the exception handling coordinator.

8.2 Restart an instance

In some cases, it is necessary to start over the whole process instance from the very beginning. The scenario of restarting a workflow instance is shown in Figure 21. Once the exception handling coordinator (EHC) gets the solution of restarting a workflow instance, it locates an appropriate Agent by checking the CPR inside the action information block of the case. Agent then obtains the workflow id and the start task specification for this workflow. When the transition call is constructed over the start task, it is immediately invoked. The workflow instance with the requested instance id is generated.

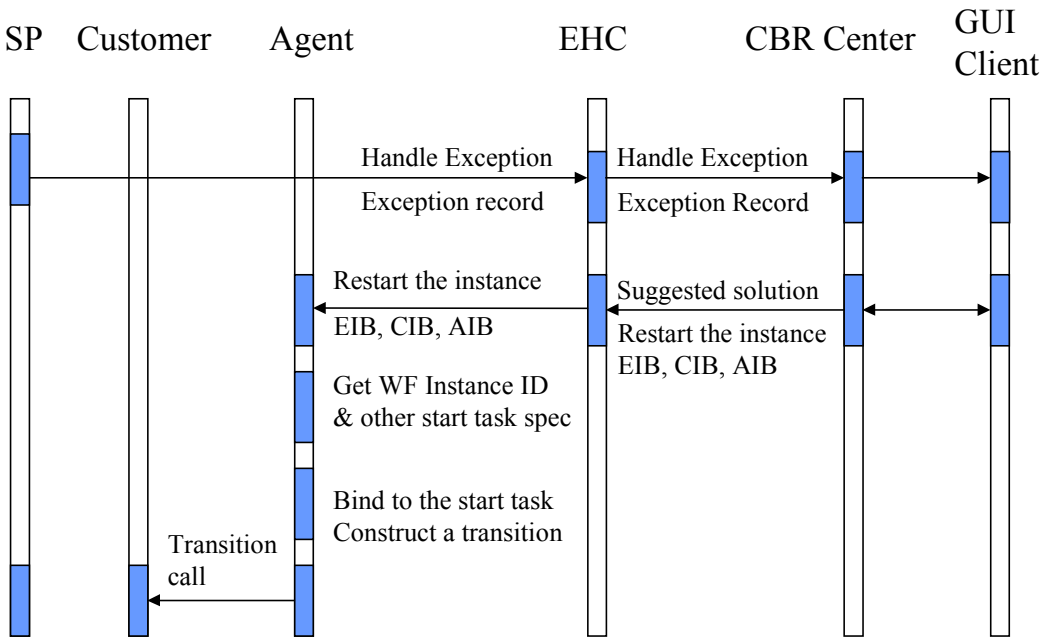

Figure 21 Restart a workflow instance

8.3 Retry and alternative task implementation

In most of the cases, it is necessary to retry the task or start an alternative task. The scenario is shown in Figure 22. Once the EHC gets the solution of retrying a task, it locates an appropriate Agent by checking the CPR inside the action information block of the case. Agent then obtains the workflow id and the task's specification for this workflow. Then the reference of this task's scheduler reference is obtained. The instance data and task parameters are also obtained by using the instance and date reference copy maintained. The values of the data will be updated by the values contained in the context information block in the case. When the transition call is constructed over the task that is to be re-activated, it is immediately invoked. The task is retried or re-started.

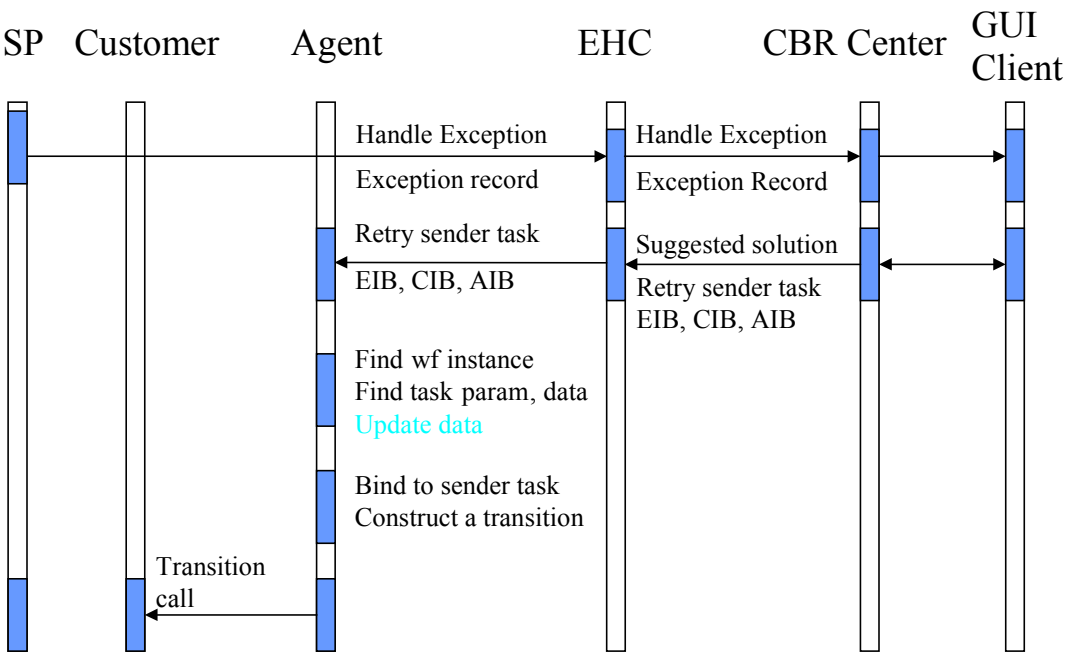

Figure 22 Retry and alternative task implementation

8.4 Exception handling mode

Two system working modes are possible - automatic mode and user intervention mode. Working mode is configurable. It can set in the cbr.properties file, a file providing configuration entries for exception handling system, which holds all the property settings to make the system work.

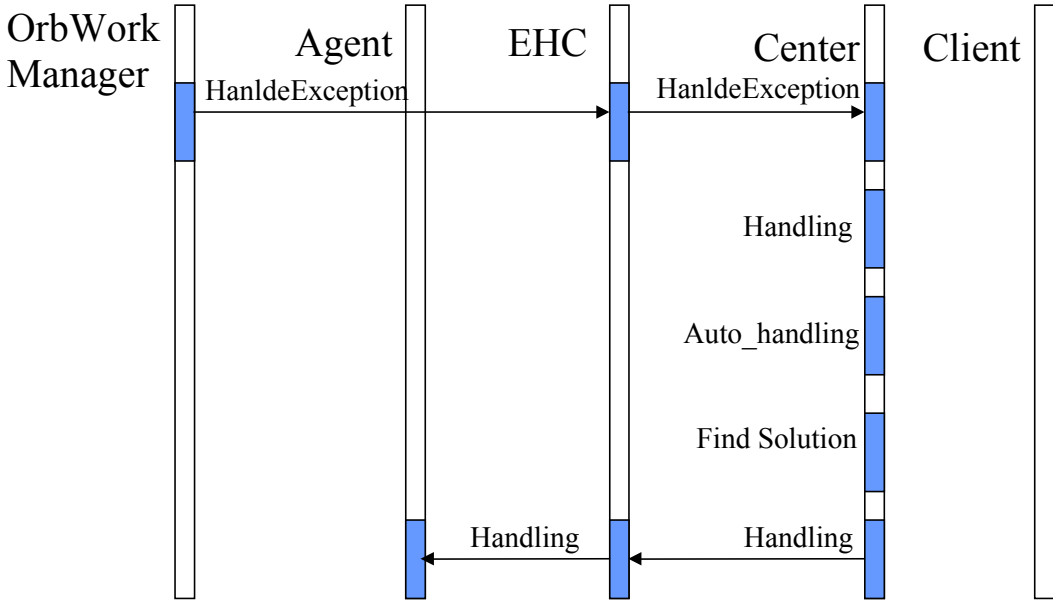

Figure 23 Automatic system working mode

8.5 Automatic mode

If the property "CBR_USER_INTERACTION" is set to 0, then the intelligent problem-solving component works in the automatic mode. In the automatic mode, the intelligent problem solver tries to find solutions

to the exception situations in an automatic way. The scenario of automatic working mode is shown in Figure 23.

First if an exception occurs, and it cannot be handled, one of the solutions is to propagate it to exception handling coordinator (EHC) through a CORBA call HandleException(). When the EHC receives the request, it finds a proper intelligent problem solver (Center), and propagates the exception to the Center. Once the Center receives the exception, it will generate an exception record to record the exceptional situation. A key is generated for this record by computing the exception type, exception message, workflow type, and component name where the exception is originally thrown, host, and time stamp. After the record is generated, the "CBR_HUMAN_INTERACTION" property contained in the cbr.properties file is checked. If it is set to zero, then the Center enters automatic handling mode. The case repository is consulted at this time. There are four level case searching priorities. The search is first conducted over the whole exception record generated. This is called exact match. If a case is retrieved, the CIB block will be checked. If the case is adaptable, which is denoted by the suffix of CPR_name, then this case will be supplied to EHC by calling a CORBA call Handling(). The EHC will locate an appropriate Agent by checking the CPR. The final solution will be transferred to Agent to execute the handling plan by calling a CORBA call Handling() on Agent. If the compensate scheme is not null, then the compensation scheme will be first executed. The rework scheme is executed after compensation scheme finishes.

If during the exact match, no case is available, Center enters the second level search. It will try a partial match over exception types, workflow types, and component name where the exception occurs in the exception record. If several cases are retrieved, a similarity measure is conducted. That process instance data will be retrieved and the values will be used to match against the case context information block. This is called context match. In the context match, this context will be used to match the contexts in the retrieved cases. The concept tree will be consulted during this stage. The case with smallest distance will be chosen as the candidate. The CPR in this case will be checked to see whether this case is adaptable. If it is adaptable, then the Handling() will be called over EHC and then Agent to deliver the solution. Otherwise, the Center enters manual mode.

If during the second level search, no cases are retrieved, the Center enters the third level search. The match will be checked over exception type and workflow type. If there are any candidates, they will be processed following what we have described in the second level search. Otherwise, fourth level search is conducted by matching exception type only.

8.6 User intervention mode

In the user intervention mode, administrators participate in the exception handling process through using a GUI based CBR client.

Similar to the automatic mode, if an exception occurs, and it cannot be handled, one of the solution is to propagate it to exception handling coordinator (EHC) through a HandleException() call. When the EHC receives the request, it finds a proper problem solver (Center), and propagates the exception to the Center.

Once the Center receives the exception, it will generate an exception record to record the exceptional situation. A key is generated for this record by computing the exception type, exception message, workflow type, and component name where the exception comes from, host, and time stamp. After the record is generated, the "CBR_HUMAN_INTERACTION" property will be checked. If this property is not set to zero, then the Center enters manual handling mode. The expert needs to use the GUI client to retrieve propagated exceptions first. Then the expert needs to search the case repository to find solution candidates. Similarly there are four level case searching priorities as in the automatic mode. The difference from automatic mode is when there is a case candidate it will be retrieved and the expert decides whether to execute the CPR scheme in this case. The expert can modify the case. Once the expert thinks the solution is workable, he can send the handling request to the EHC and Agent later via the GUI client. The expert can also decide to write the new case into the case repository.

9. Exception Handling Example

In this section, an exception-handling scenario is demonstrated to illustrate the exception handling mechanism presented so far. In this scenario, we will generate an exception. This exception will then be propagated to the exception handling system. This exception will be generated by removing a task from OrbixWeb's implementation repository [24]. The main purpose of the demonstration is to show how a case adapts to a new situation, i.e., how a case is reused.

As described previously, a case consists of three blocks: EIB, CIB, and AIB. Except for the content of EIB, attributes in CIB and AIB can be modified to adapt to new situations. This is what we call case reuse - obtaining a new case for the new situation by changing a previous case. At this time, this technique is used to compact the case repository, thus enhancing the efficiency of the exception handling system.

In this demonstration, we show how the handling scheme stored in the case repository is modified to obtain a new case. This type of case reuse is called parameter based case adaptation. When a case is used for a new situation without modifications, it is usually called NULL adaptation.

In this demonstration, we first build the level 3 application, compile it, and then deploy it. Inside the level 3 application (see Figure 1), there is a task called check_identity that will update the database with a record. We then remove this task from OrbixWeb's implementation repository. An exception will be thrown because this task can not be found by the OrbixWeb daemon. Then it will be propagated to the exception handling coordinator.

The following are the steps of this scenario:

1. Deploy the level 3 workflow application.
2. Create a new instance for this application.
3. In the workflow application, there is a task called check_identity. Remove the task of check_identity from OrbixWeb's implementation repository.

4. When the exception is propagated to the exception handling coordinator, which can be verified by looking into the monitor file or the exception handling coordinator execution trace, launch the exception handling client GUI tool (see below).
5. First make sure the exception-handling client is connected to the exception handling coordinator. If yes, then retrieve the exception record propagated by clicking "Retrieve" button. When a record appears in the exception retrieval panel, click on that record, and then click the button "Handle".
6. Go to the case search panel, click "Search" button. Since the case repository does not contain any similar cases, a case template will be generated. Click on the case, the click "Analyze" button to analyze the case.
7. Go the case analysis panel, make the following changes to the case:
   Compensation_mode: auto
   Compensation_type: exception_handler
   Conpensation_action: register_server
   Number of param: 2
   Workflow: l3
   Task name: check_identity
   Rework_mode: auto
   Rework_type: restart
   Rework_task_name: check_identity
   Rework_host: mitchell.cs.uga.edu
   CRP_name: _ehc_wft
8. Now click the "Action" button. Then go to the take action panel, click the button "Take Action".
9. It can be noticed that the result that the task of check_identity is registered, and it is re-restarted.
10. Save this case into the repository by clicking on the button "Write back".
11. Uninstall the level 3 application.
12. Re-install the level 3 application. Remove the check_indentity task from OrbixWeb's implementation repository.
13. Create a new instance of level 3 application. Then finish the task of sender, you will then see that an exception is propagated to the exception handling coordinator and handled automatically by retrieving this previous captured case. This type of case reuse is called NULL case adaptation because the case is not modified.
14. Uninstall the level 3 application.
15. Re-install the level 3 application. Remove a different task called "connection_request" from OrbixWeb's implementation repository.

16. Create a new instance of level 3 application. Then finish the task of sender, you will then see that an exception is propagated to the exception handling coordinator and handled automatically by retrieving this previous captured case. Since that case contains solution for handling solutions for the task of "check_identity", it must be modified to adapt this new situation. So the task name in this case is modified from "check_identity" to "connection_request", the task state string and task parameter string are modified accordingly. Thus, a new case is created by reusing a previous acquired case. This type of case reuse is called parameter based case adaptation. When the whole block of CIB and/or AIB are modified, which means a new CIB and/or AIB are used, it is called substitution based case adaptation.

10. Approach Evaluation

The essence of CBR is that similar problems can be solved by similar solutions. In the following, we will show our evaluation of the applicability of CBR in our exception handling system. This evaluation is conducted upon the analysis of CBR mechanism. There are two assumptions upon which CBR is based. One is that problems tend to re-occur. The other assumption is that solutions are applicable for similar problems. Therefore, to prove the applicability of CBR in exception handling, it is necessary to show that (1) problems in cross-organizational business processes tend to re-occur, and (2) solutions to previous problems can be applied to similar re-occurring problems.

To evaluate the applicability of CBR in exception handling, we have conducted two experiments. We conduct the first experiment to test the problem re-occurrence rate, and the second experiment to find out the distribution of problem re-occurrences. Moreover, we give an analysis about the relationship between encountered problems and their solutions. In experiment 1, we build the Level 3 telecommunication workflow application, and execute it as usual. Then we do nothing to the occurring exceptions but record them. There are total 133 exception records. To calculate the re-occurrence rate, we classify the exceptions according to their types. Among these exceptions, we have found that each exception occurred at least twice in one run. Majority of these exceptions' re-occurrence rate is about 17 per run. In experiment 2, we build another application, and execute it as usual. Again, we do nothing to the occurring exceptions but record them. There are total 159 exception records. This newly built application contributes 27 additional exceptions. To understand these exceptions, we classify the exceptions according to their types and workflow applications. Among these exceptions, we have found that each exception occurred at least twice in one run. We have also found that majority of these exceptions' re-occurrence rate depends on the number of tasks in workflow applications. This founding is very interesting. By conducting this simple histogram analysis of exceptions, we have obtained several useful insights about these exceptions:

- If an exception only occurs in one application, but not others, it is usually caused by application dependent problems.

- If an exception occurs in every application, it is usually caused by problems in the workflow management system components.
- If the re-occurrence rate of an exception is low, this exception is usually caused by application dependent problems.
- If the re-occurrence rate of an exception is high, this exception is usually caused by workflow system problems.

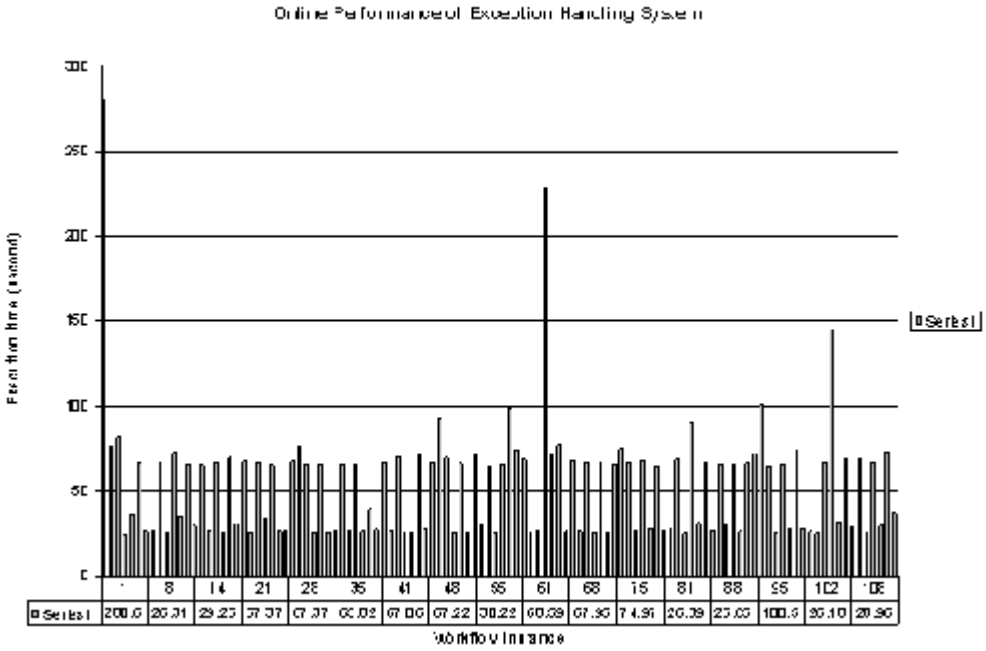

Figure 24 Execution time of the handling exceptions

CBR, which is used to find these kinds of exception patterns, will add great value to the exception handling coordinator. Now we will describe our experiment by using the L3 telecommunication application to get performance evaluation of the exception handling system. We test how many workflow instances can be running with exception handling system always participating in the workflow execution. That is, for each instance, there is an exception occurring. Exception handling system is participating in each workflow instance execution. To generate an exception that can be handled by the ORBWork system, we supply an incorrect or inappropriate statement to the database systems (we are using Mini-SQL database server). An exception of "database statement has no result" will be thrown.

In this test, we were able to generate about 118 workflow instances (see Figure 21) with currently running exception handling system. It seemed that we were able to generate more instances, but the system was quite slow. The average exception resolution time is about 54 seconds. The exception resolution time includes record generation time, case retrieval time, case analysis time, and CPR execution time. We can see that the execution time for several workflow instances is quite large. The reason is that because the

system is first started, a lot of time needs to be spent on the system preparation, such as object initializations.

11. Summaries and Future Work

Compared to extensive prior work on exception handling in programming languages and distributed system, cross-organizational processes present new challenges. Some of these challenges are the need to determine responsible party for handling exceptions, a variety of differences between exception handling mechanisms of each WfMS participating in cross-organizational processes, and lack of understanding or knowledge of outsourced or contracted processes.

In this paper, we have presented our research on cross-organizational workflow exception handling based on our understanding of the cross-organizational business processes. An advanced exception handling mechanism is proposed for handling exception across organizational boundaries. This exception handling technique bundles three techniques, exception-handling knowledge sharing, coordinated exception handling, and intelligent problem solving.

The novel research contributions presented include:

- Application of case-based reasoning (CBR) in exceptional problem solving for cross-organizational business processes. Although CBR is used in handling exceptions inside a WfMS before, it has not been used in a cross-organizational setting in the past.
- Use of a similarity-matching scheme in the CBR that includes exception, process, and context matching in the case matching for handling exceptions across organizational boundaries. In particular, we support partial match to identify relevant cases.
- Process dynamics exploration for the construction of flexible exception handling processes across organizational boundaries. Earlier work on dynamic workflows (flexible processes) in cross-organizational setting considers various process modes but do not discuss exception handling.
- A bundling strategy that provides more powerful solution than each of the individual techniques of knowledge sharing, coordination, and intelligent problem solving.

We have built several workflow applications to test our exception handling system. The effectiveness of our exception handling systems is strongly related to the flexibility of the used WfMSs. If a specific WfMS is weak in adapting to the changing environment, our exception handling system will also be limited in adaptation. In such a case, though our exception handling system can propagate exceptions across organizational boundaries, the actual exception handling schemes (CPR) can only be used for the exact same situations encountered before.

Future directions for this work are as follows:

- Improving the system by learning from the exceptions. Once the cases have been obtained, we still need criteria for when and how the system can be improved based on what we have learned.

- Construction of processes based on the cases obtained. Case repository can hold hundreds of cases. Besides using these experiences for handling exceptions, it is also very interesting to construct new processes by using the cases obtained.
- Survivable computing. Current research is being conducted in LSDIS Lab to use this exception handling system as a core component to enhance the survivability of workflow systems [25]. Since dynamic changes are considered as exception handlers, the system built in this paper couples two key capabilities, exception handling and dynamic changes to support survivability along with techniques from the fault-tolerant computing.

References

[1] H. Ludwig and K. Whittingham, "Virtual Enterprise Coordinator - Agreement-Driven Gateways for Cross-Organizational Workflow Management", in Proceedings of the International Joint Conference on Work Activities coordination and Collaboration, WACC'99, February , 1999, San Francisco, CA.

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

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

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

[5] H. Saastamoinen, "On the Handling of Exceptions in Information Systems". PhD Thesis, University of Jyvaskyla, 1995.

[6] M. Klein et al., Proceedings of CSCW-98 Workshop Towards Adaptive Workflow Systems, Seattle, WA, 1998.

[7] F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi, "Specification and Implementation of Exceptions in Workflow Management Systems", TODS, Vol 24, No. 3, pp.405-451, 1999.

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

[9] P. Berry, K. Myers; Adaptive Process Management: An AI Perspective, CSCW, Towards Adaptive Workflow, Seattle, WA 1998.

[10] W. Deiters, T. Goesmann, K. Just-Hahn, T. Loffeler, and R. Rolles, "Support for Exception Handling through Workflow Management Systems", CSCW, Towards Adaptive Workflow, Seattle, WA 1998.

[11] Z. Luo, Amit Sheth, Krys Kochut, and John Miller, "Exception Handling in Workflow Systems", Applied Intelligence: the International Journal of AI, Neural Networks, and Complex Problem-Solving Technologies, Volume 14, Number 2, September/October, 2000.

[12] M. Klein and C. Dellarocas, “A Knowledge-Based Approach to Handling Exceptions in Workflow Systems”, Proceedings of CSCW-98 Workshop Towards Adaptive Workflow Systems, Seattle, WA, 1998.

[13] M. Klein, “Towards A Systematic Repository of Knowledge about Managing Multi-Agent System Exceptions”, Working Paper, ASES-WP-2000-01, Center for Coordination Science, MIT, Cambridge, MA February 2000.

[14] CAISE Workshop on Infrastructures for Dynamic Business-to-Business Service Outsourcing, Stockholm, June 5-6 2000.

[15] H. Ludig, C. Bussler, M. Shan, and P. Grefen, “Cross-Organizational Workflow Management and Co-ordination Workshop Report, February 23nd 1999, San Francisco, CA.

[16] H. Ludwig, "Termination Handling in Inter-organisational Workflows - An Exception Management Approach", In A. Antola (Ed): *Proceedings of the Seventh Euromicro Workshop on Parallel and Distributed Processing (PDP '99)*, Funchal, February 1999, pages 133 - 139, IEEE Computer Society, Los Alamitos, 1999.

[17] Z. Luo, “Knowledge Sharing, Coordinated Exception Handling, and Intelligent Problem Solving for Cross-Organizational Business Processes”, Ph.D. Dissertation, Department of Computer Science, University of Georgia, Athens, January 19, 2001.

[18] K. Kochut, "METEOR model 3", LSDIS Memorandum, Department of Computer Science, The University of Georgia, 2000.

[19] L. Bittel, "Management by Exception; Systematizing and Simplifying the Managerial Job", New York, McGraw-Hill, 1964.

[20] "Harvard Business Review on Knowledge Management", Boston, MA: Harvard Business School Press, 1998.

[21] I. Arpinar, J. Miller, and A. Sheth, “An Efficient Data Extraction and Storage Utility For XML Documents”, 39th Annual ACM Southeast Conference, Athens, GA, March 2001.

[22] A. Aamodt, “Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches”, Artificial Intelligence Communications, IOS Press, Vol. 7: 1, 1994 .

[23] K. Kochut, A. Sheth, and J. Miller, "Optimizing Workflow, Using a CORBA based, Fully Distributed process to Create Scalable Dynamic Systems", Component Strategies, March 1999, pp. 45-57.

[24] http://www.iona.com/docs/orbixweb/orbixweb32se.html

[25] J. Cardoso, Z. Luo, J. Miller, A. Sheth, and K. Kochut, “Survivability Architecture for Workflow Systems”, 39th Annual ACM Southeast Conference, Athens, GA, March 2001.