Knowledge Sharing, Coordinated Exception Handling, and Intelligent Problem Solving for Cross-Organizational Business Processes

PhD Dissertation

ZongWei Luo

January 19, 2001

Major advisor:
Professor Amit Sheth

Committee members:
Professor Krys Kochut
Professor John Miller
Professor Annette Poulsen
Professor Richard Watson

Invited committee member:
Dr. Christopher Bussler

Large Scale Distributed Information Systems Lab
Computer Science Department, The University of Georgia

KNOWLEDGE SHARING, COORDINATED EXCEPTION HANDLING, AND INTELLIGENT PROBLEM SOLVING FOR CROSS-ORGANIZATIONAL BUSINESS PROCESSES

Zongwei Luo
Under the Direction of Amit Sheth

To date, Workflow Management Systems (WfMSs) have been used to support enterprise-wide business processes. With the advent of Internet commerce, business processes increasingly span organizational boundaries. Consequently, workflow technology needs to be extended to support such cross-organizational processes. Three of the most important research issues that arise in developing solutions for cross-organizational business processes are -- process construction, service fulfillment, and conflict resolution. This research primarily addresses the challenge of conflict resolution in cross-organizational processes using exception-handling techniques.

Compared to extensive prior work on exception handling in programming languages and distributed system, cross-organizational processes present new challenges. Key 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.

This dissertation represents one of the earliest comprehensive researches on the topic of conflict resolution in cross-organizational processes. We have proposed a detailed exception-handling strategy, have implemented a prototype system, and have conducted experiments using realistic applications to test its feasibility using a working WfMS. Our exception handling mechanism bundles knowledge sharing, flexible process coordination, and intelligent problem solving to handle exceptions in cross-organizational settings.

Novel research contributions presented here include:
1. 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 cross-organizational setting in the past.
2. 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.
3. Process dynamics exploration for the construction of flexible exception handling processes across organizational boundaries. Earlier work on dynamic workflow (flexible process) in cross-organizational setting considers various process modes but do not discuss exception handling.
4. A bundled strategy that provides more powerful solution than each of the individual techniques of knowledge sharing, coordination, and intelligent problem solving.

INDEX WORDS: Knowledge sharing, Exception handling, Intelligent systems, Business processes, Workflow, Inter-enterprise

TABLE OF CONTENTS

KNOWLEDGE SHARING, COORDINATED EXCEPTION HANDLING, AND INTELLIGENT PROBLEM SOLVING FOR CROSS-ORGANIZATIONAL BUSINESS PROCESSES

ZONGWEI LUO

ACKNOWLEDGEMENTS

This research project is part of the group effort for the METEOR Workflow research at the Large Scale Distributed Information System Laboratory (LSDIS) of the Computer Science Department at the University of Georgia. First I wish to thank my major advisor Dr. Amit Sheth for supporting me for my whole Ph.D. study. This support is crucial to me to make this dissertation happen. Thank you very much! I also wish to thank Dr. John Miller and Dr. Krys Kochut for your advice, encouragement and help. Without them, it would have been impossible for me to finish this dissertation.

Special thanks should go to Dr. Annette Poulsen and Dr. Richard Watson for their kindness of participating in my committee and all the advice and help you gave me. Special thanks also go to Dr. Christopher Bussler for helping me pass the written comprehensive examination. Without them, it would have been impossible for me to finish my Ph.D. study.

I also wish to thank Dr. Myong Kang for the help he gave me and the generous support he gave for the METEOR project.

In the mean time, I wish to thank my colleagues in the LSDIS Lab, Devashish Worah, Zhongqiao Li, Jorge Cardoso, and Budak Arpinar. I appreciate your help and collaboration!
Thank you all!

ZongWei Luo
January 19, 2001

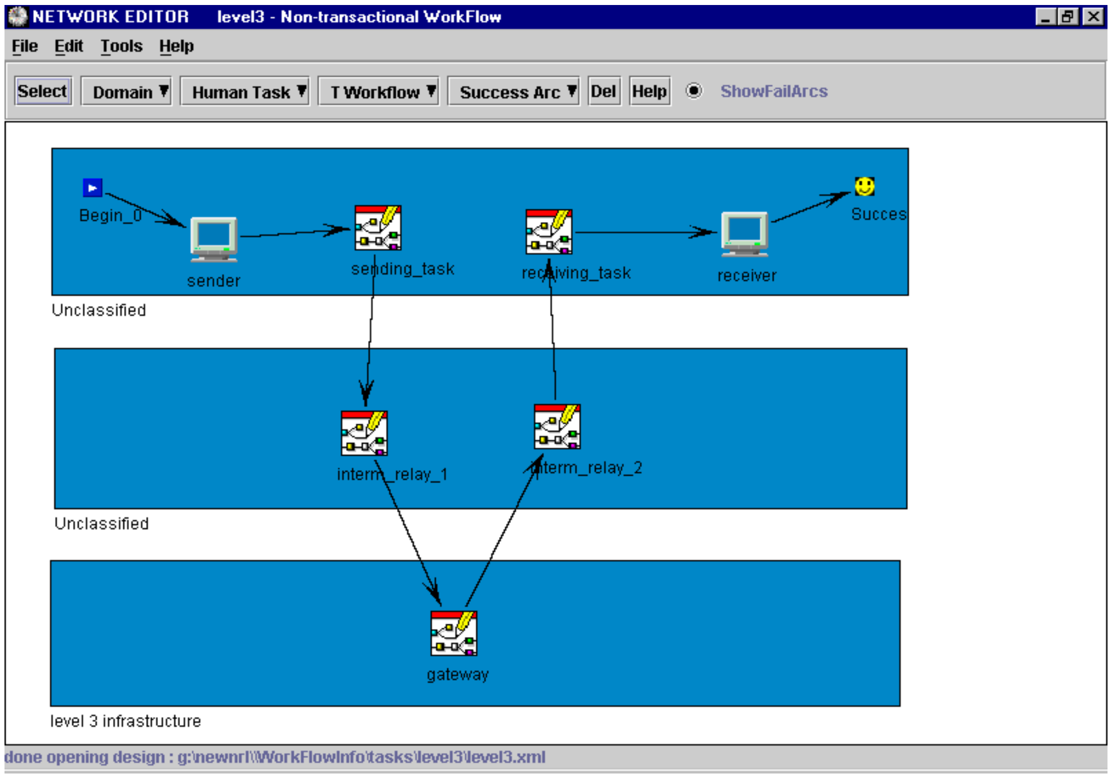
CHAPTER 1

INTRODUCTION

Until several years ago, electronic trading was primarily involved with expensive and specialized processes. This has hindered many businesses, especially small businesses from participating in the electronic trading. Over the last few years, the Internet has matured to provide a global networking infrastructure, accessible to a huge and yet increasing number of business organizations and people [Forrester]. It has transformed electronic trading into a viable and cost effective business solution: Internet trading. Increased ease of Internet trading provides opportunities for organizations to work more closely with other organizations for the exchange of service, goods and information. Internet helps organizations establish intimate business relationships with their customers and material suppliers [Forrester]. Through participating in such electronic trading activities, these organizations can focus on their core business, while lowering operational costs. Internet has also been utilized to improve customer service. For example, web portals have been built as one-stop shopping site for the customers. Online interactive-help systems have been built to serve as help desk for the customers.

BUSINESS PROCESSES

The hyper-growth of Internet commerce affect businesses by making trading processes more efficient and less expensive. There are usually three distinct types of intermediate business trading processes: auction, bid, and catalog [Adam et al 1999]. Business entities and processes interact with each other through these intermediate-trading processes, thus creating a dynamic trading process. These business-trading processes, as an organic part of doing business, are being deeply integrated as a critical component of almost all types of systems that support business activities [Sheth et al 1999]. 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 these business processes. In Internet commerce, business processes involving business-to-business and business-to-customer activities usually span across multiple enterprises. This requires that Workflow Management Systems (WfMSs) provide a set of tools supporting the necessary services of workflow creation, workflow enactment, and administration and monitoring of these business processes in cross-organizational settings. One promising technique to realize such a support for cross-organizational business processes is process outsourcing, usually through contracting [Ludwig and Whittingham 1999].



done opening design : g:\newnrl\\WorkFlowInfo\tasks\level3\level3.xml

Figure 1.1 A Cross-organizational workflow: telecommunication bandwidth outsourcing

The 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. It partly results from the telecommunication sector deregulation. Many telecommunication infrastructure providers often contract out their unused bandwidth to other telecommunication service providers. To facilitate the bandwidth outsourcing, cross-organizational processes are used.

An example of such cross-organizational processes in telecommunication sector is shown in Figure 1.1. In this example, a telecommunication infrastructure provider, Level 3 (L3), has designed an all-IP network that is scalable and upgrade-able. It offers on-demand bandwidth service to let contracted service providers (SP), such as Internet Service Providers (ISP) and DSL providers, accommodate their customers' new data and voice applications. The interfaces to L3's network infrastructure are called Gateways. These gateway facilities provide 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, soft-switches and advanced security to allow interconnections with other local and long distance networks. L3's direct customers are other service providers (SP). 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. In case the request is rejected, the rejection response must come within one minute. This requirement is usually specified in the Service Level Agreement (SLA) between L3 and these SPs. The customers subscribe to use L3's infrastructure through these SPs (see Figure 1.2). When a subscribed customer requests a connection to a third party, his SP has to route it to the Gateway through a local network carrier. To do so, these SPs need to sign contracts with local network carriers to reach their customers and route their customers' connections. If the customers' connection requests exceed their SPs' contracted bandwidth, the SPs can request bandwidth upgrade to accommodate their customers' needs.

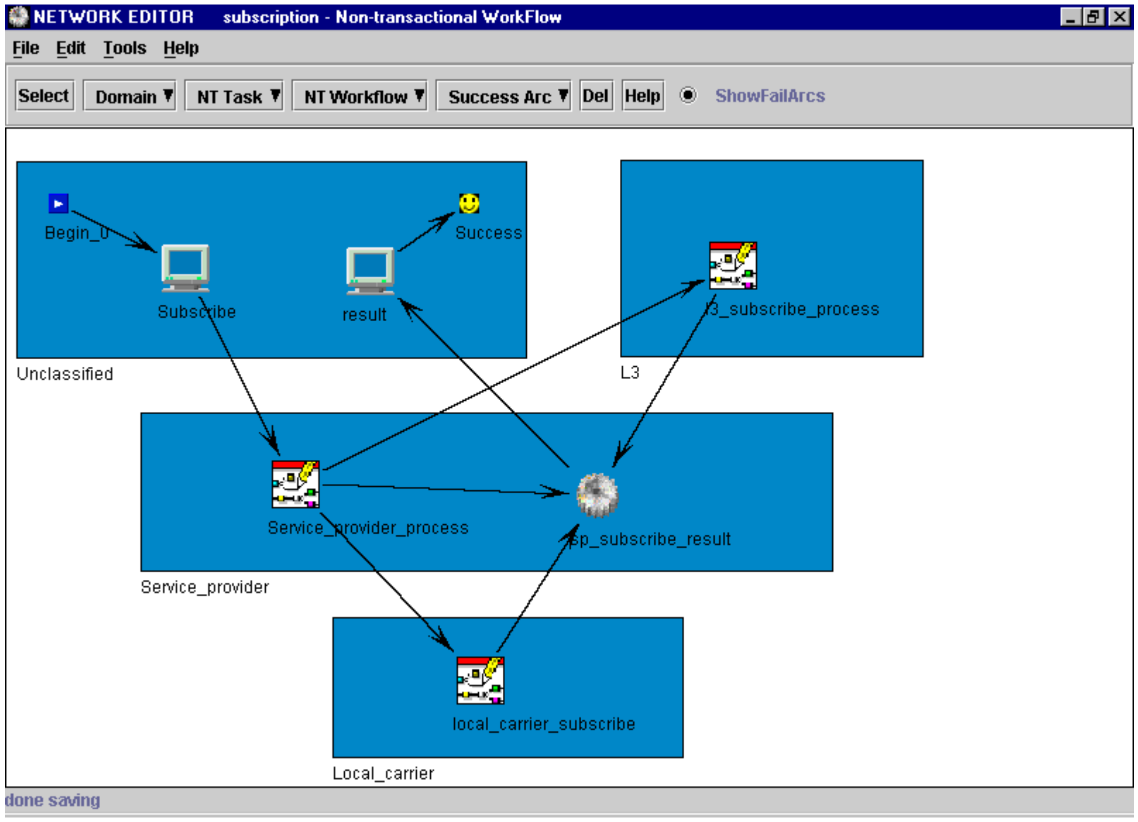


Figure 1.2 A cross-organization workflow: customer connection request process

TARGETED PROBLEMS

One major difference between cross-organizational and enterprise-wide business processes is that cross-organizational business processes are realized by cooperating processes across organizational boundaries. Contracts are set up so that these processes should meet these contracted cooperating requirements. However, abnormal situations may occur, for example, when the cooperating requirements can not be met. Abnormal situations in cross-organizational processes are classified as follows.

- A contract cannot be fulfilled. For example, the bandwidth change service routed by SPs can not be fulfilled by L3; therefore, SPs' customers will get busy signal due to traffic congestion.
- A contract may be compromised. For example, rejection response to the customer's bandwidth change can't always happen within a minute. L3 asks contracted SPs to allow the rejection response be given longer than a minute.
- A contract needs to be modified. Due to sustained customer growth, rejection responses to the customer's bandwidth change can no long happen within one minute. L3 negotiates with contracted SPs to change the contract.
- A contract needs to be terminated before it expires. When L3 fails to fulfill its obligation too often and a contracted SP may consider terminating its contract before it expires.

In case any of these abnormal situations occur, they must be resolved so that the outsourcing partnership can be maintained in the effort that both parties can benefit from the outsourcing partnership. An example of such a conflict (exception) in this telecommunication application is that the rejection response does not come within one minute after the requested service is rejected. When this conflict (exception) occurs, processes that span more than one organization are affected. We have identified the problems in resolving such exceptions across organizational boundaries. They are summarized in Table 1.1.

Table 1.1 Exception handling problems in cross-organizational settings

	Problems
Specification	Some exceptions defined in one organization are not defined in another organization.
	The exception format may be different in different organizations
	Some exceptions have different semantics or interpretations in different organizations.
Propagation	Some exceptions are local to an organization, even if they are related to another organization, which is unaware of its occurrence.
	The exception propagation path is not well defined in cooperating organizations.
	When an exception caused by one organization is detected by another organization, it is possible that not enough information is available to verify it.
Handling	Different organization may have different understanding of an exception. Thus the handling policy may be different.

Healthcare sector is another important field for studying cross-organizational business processes, especially for exception handling. Let’s take a look at an infant healthcare application to characterize the problems in exception handling (see Figure 1.3). This infant transportation application involves the transportation of a very low birth weight infant of less than 750 grams, at or below 25-26 weeks gestation, from a rural hospital located within 100 miles from the Neonatal Intensive Care Unit (NICU) at the Medical College of Georgia (MCG). Such transportation usually takes up to 2.5 hours. In the ambulance there are two or three healthcare professionals who perform different roles. During the transport the ambulance personnel perform standard procedures to obtain medical data for the infant. This application involves three organization domains, local hospital, ambulance, and NICU. The first step of this application involves the task that allocates resources in the local hospital, such as healthcare professionals, equipment, and so on. The last step is admission preparation to the NICU.

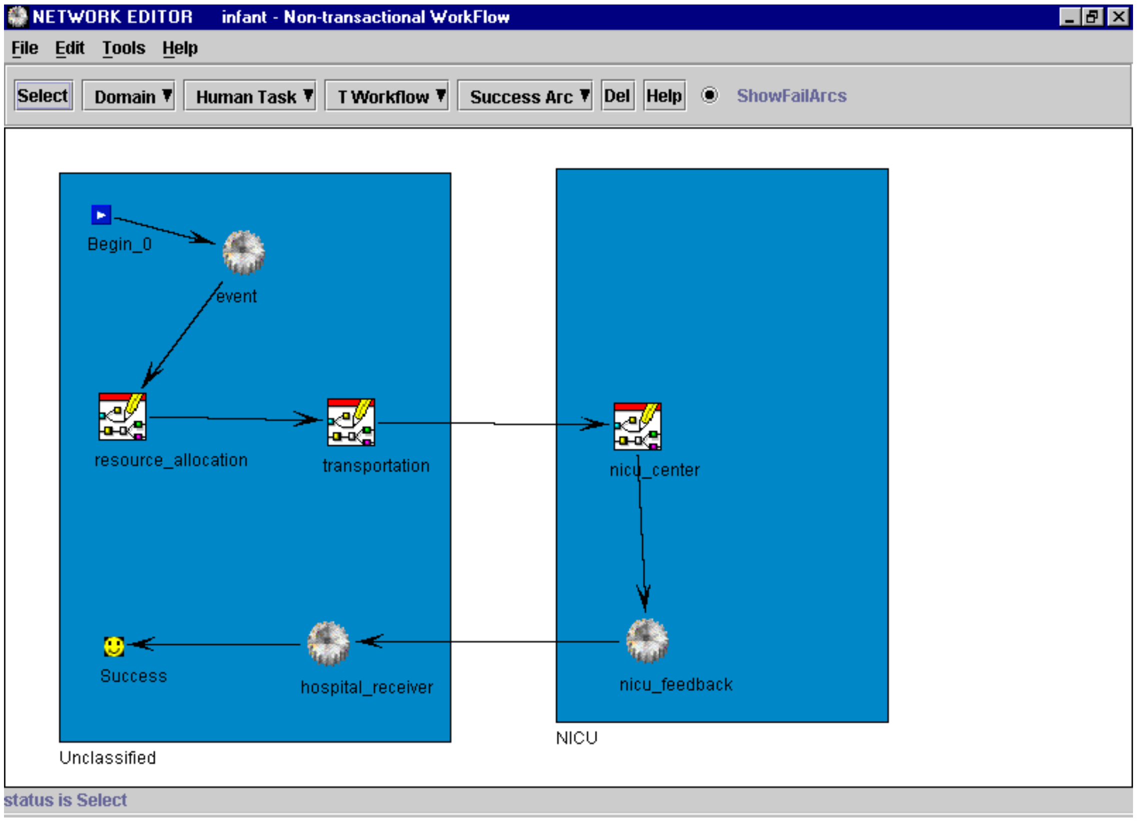


Figure 1.3 Infant transport healthcare application

If exceptions are raised during the transport, corrective actions must take place. The decision of the corrective procedures usually involves collaboration and coordination between the ambulance personnel and the consultants at MCG's NICU. This application is very dynamic because the changes to the infant's health status as indicated by the vital signs such as the known risk factors may lead to changes in the treatment plan. Such changes can occur rapidly. For example, a low weight infant can dehydrate in as few as ten minutes while an adult would take at least several hours to reach the same severity. Such changes to the infant's status are modeled as exceptions. Consider a "normal" treatment plan as shown in Figure 1.4. Occurrence of heart murmur that is known as a risk factor related to cardiac disease would be modeled as an exception (from the normally expected and correspondingly modeled process). One way of handling such an exception is to change the process such that the cardiovascular related task is performed earlier than what was originally planned.

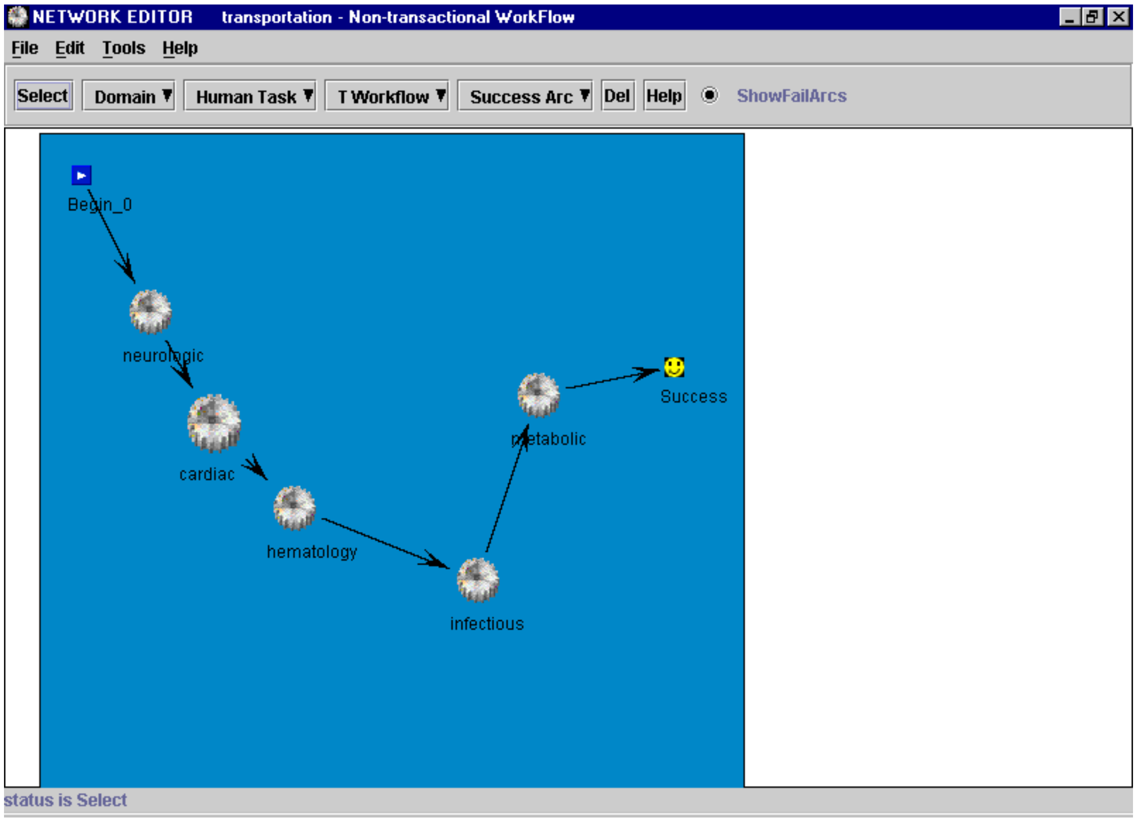


Figure 1.4 Treatment cares during the transportation

This healthcare application raises several requirements for business process support in cross-organizational settings.

- Exceptions are not avoidable in such an environment. An abnormal situation can cause special attention for healthcare professionals. Those abnormal situations should be resolved as soon as possible due to the nature of this application -- newborn infant transport. Prior experience gained in handling similar abnormal situations can facilitate the exception resolution process. At the same time the set of exception handlers that need to be checked, can be limited by capturing more process execution context.

- The processes in this application are very dynamic. The events that drive the dynamic change and evolution may often come from different organizations. In many cases, the driven forces are exceptional situations. Because these organizations often have contracts with each other, they are also affected by such exceptional situations in other organizations. To support coordination of such processes, a systematic way of workflow evolution, including dynamic structure modifications should be worked out. In our work, they are considered as possible exception handler candidates.
- To resolve exceptional situations, potential exception handling activities may happen across organizational boundaries in this transport process, e.g., the healthcare professionals on board may need advice from specialists in the NICU. The advice is context based. The results from the information exchange may affect the progress of the ongoing exception handling process.
- The health professionals on board are not necessarily experienced in every aspect of intensive care. A case repository used in the case-based reasoning (CBR) based exception-handling system stores valuable experience learned to help them make decisions. The experience should be available and accessible to health professionals from different organizations.

In this dissertation, we discuss a bundled solution to address the problems in cross-organizational exception handling activities, such as heterogeneity, responsibility determination, and black box effects. There are three major objectives in this approach for exception handling in cross-organizational settings. The bundled solution achieving these three objectives connects the following three components organically, thus forming a sophisticated exception handling framework for handling exceptions in cross-organizational settings.

- The first objective is to capture more context information and gain common understanding in exception handling. By capturing more context information of exception situations, possible alternative exception handlers can be limited in dealing with exceptional situations. By reaching more common understanding and sharing exception handling experience, exception resolution processes can be facilitated. The exception handling knowledge stored in a case repository can be shared among different organizations.
- The second objective is to understand the coordination challenges in the exception handling process across organization boundaries. Business processes belong to different organizations. An organization may have the rights to access processes in other organizations. But this does not mean that it has the necessary right to fully control them as it does to its own processes. For example, workflow evolution, a good candidate to exception handling, is not always available due to such organizational constraints. That is, possible process modifications, along with other exception handler candidates such as ignore, retry, workflow recovery, and so on, may behave differently in cross-organizational exception handling processes.
- The third objective is to find a feasible intelligent problem solving technique in resolving exceptional situations. Exceptional situations are generally very

complicated. Intelligent problem solving capability is usually a must for an advanced exception handling system. In our work, a case-based reasoning (CBR) based exception handling mechanism with integrated human involvement is used to support exception-handling processes. This mechanism enhances the exception handling capabilities through collecting cases to capture experiences in handling exceptions, retrieving similar prior exception handling cases, and reusing the exception handling experiences captured in those cases in new situations. As stated in the first objective, the exception handling knowledge is shared among organizations; the actual case users are not necessarily the ones who have collected these cases.

CONTRIBUTIONS

This dissertation represents one of the earliest comprehensive researches on the topic of conflict resolution in cross-organizational processes. We have proposed a detailed exception-handling strategy, have implemented a prototype system, and have conducted experiments using realistic applications to test its feasibility using a working WfMS. Our exception handling mechanism bundles knowledge sharing, flexible process coordination, and intelligent problem solving to handle exceptions in cross-organizational settings.

Novel research contributions presented here include:

1. 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 cross-organizational setting in the past.
2. 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.

3. Process dynamics exploration for the construction of flexible exception handling processes across organizational boundaries. Earlier work on dynamic workflow (flexible process) in cross-organizational setting considers various process modes but do not discuss exception handling.
4. A bundling strategy that provides more powerful solution than each of the individual techniques of knowledge sharing, coordination, and intelligent problem solving.

Organization of this dissertation is as follows. In Chapter 2, we review related work. In Chapter 3, we present our approach for modeling cross-organizational business processes, our analysis of process dynamics, and our strategy of exception handling. In Chapter 4, we discuss exceptions in cross-organizational business processes. In Chapter 5, we present our approach of exception handling knowledge sharing. In Chapter 6, we present our approach of process coordination for constructing flexible exception handling processes in cross-organizational settings. We also present our analysis of exception handling requirement in each of the process coordination patterns. In Chapter 7, we present our approach of CBR based intelligent problem solving. In Chapter 8, we discuss ORBWork runtime system. In Chapter 9, we present our exception handling system and its implementation. In Chapter 10, we present our analytic tool for analyzing CBR based exception handling system, impact analysis, and experimental analysis. We also conclude this dissertation and present possible future work in Chapter 10.

CHAPTER 2

REVIEW OF LITERATURE

Exception handling for business processes in cross-organizational settings is a multi-disciplinary area. It involves almost all aspects of process definition, enactment (including process interaction), monitoring, and administration. In this chapter, we identify the research efforts and highlight the representative research results achieved in the following four areas: (1) workflow exception handling, (2) adaptive workflow, (3) advanced transaction models, and (4) cross-organizational business processes.

- Workflow exception handling. Workflow exception handling involves exception masking and propagation. Many techniques have been proposed to handle exceptional situations in workflows. To handle exceptions, many classifications of exceptions have been made. Usually the exception handling schemes are affected by the exception classifications. For example, exceptions are usually classified in current literatures into either expected or unexpected exceptions. For those unexpected exceptions, it is claimed that they can only be handled with human involvement.
- Adaptive workflow. Due to foreseen and unforeseen situations workflow management systems needs to adapt to the dynamic and uncertain business environment. Among various situations that trigger workflow dynamic modifications and evolution, exceptions are a major source, which directly points where the WfMSs or workflow applications need to be improved.

- Advanced transaction models. Their origin is related to database technology. They are proposed to improve efficiency in long running transactions. To deal with the problems in long running business processes, people have tried to borrow this technology and tried to apply them to support workflow transactions. In this context, the term of transactional workflow was coined. The ideas of transactional workflow are workflows should be able to recover and proceed in case of failures. Most important techniques to support such workflow recovery and proceeding are backward recovery, forward recovery, and compensation.
- Cross-organizational business processes. Facilitated by the Internet, business processes deployed in various organizations have been inter-connected to form process chains across organizational boundaries. Sophisticated exception handling capability is a must for process coordination systems to support such process chains. Research activities have been reported on some basic issues of supporting cross-organization business processes, such as contracting.

EXCEPTION HANDLING

Researchers and projects (e.g., METEOR [Krishnakumer and Sheth 1995], [Luo et al 2000], and WAMO [Eder and Liebhart 1998]) 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 [Saastamoinen1995]. The author presented a theoretical basis, based on Petri-Net, for dealing with different types of exceptions. This work is purely driven by organizational semantics rather than by a workflow process model. In [Klein et al 1998], several work was presented on workflow exception handling, which included using Event Condition Action (ECA) rules to model expected exceptions, a general discussion about exceptions in systems based on objected oriented databases. In [Klein et al 1998], a taxonomy for exceptions in workflow systems was reported. This exception taxonomy combined with the exception design pattern [Casati 1999] can be reused in our intelligent exception-handling system to help measure the similarities among cases during case retrieval and analysis. Since exceptional situations are often very complicated, the knowledge-based approach is a good candidate for handling exceptions in an intelligent way by using the methodologies developed in knowledge-based systems.

In [Hagen and Alonso 1998] an exception handling mechanism employing a combination of programming language concepts and transaction processing techniques was proposed. However, aborting or canceling a workflow task would not always be appropriate or necessary in a workflow environment. Unlike a database transaction, tasks in workflow systems could encapsulate diverse operations. The nature of the business process can tolerate some errors so that an undo operation is not always required. Therefore, the error handling semantics of traditional transactional processing systems are too rigid for exception handling in workflow systems.

EXCEPTION HANDLING AND ADAPTIVE WORKFLOW

The work reported in [Berry, Myers 1998] explored how techniques from intelligent reactive control could be leveraged to provide adaptable capabilities within workflow technologies. The main artificial intelligence technologies used are agent planning and procedural reasoning.

In [Deiters et al 1998], the authors differentiated two classes of exceptions, known and unknown. To improve the business processes, they identified four perspectives: incompleteness, informal aspects, and 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. A process designer could further improve the process by changing the process, e.g., from structured process to semi-structured process to cope with the exceptions identified.

In [Luo et al 2000], 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 exception handler candidates.

KNOWLEDGE BASED EXCEPTION HANDLING

In [Deiters et al 1998], the authors offered a flexible exception handling mechanism and moved in the direction to take a knowledge-based approach in dealing with exceptions. They agreed that case based reasoning systems could be an appropriate support in the usage of knowledge base. However, in [Deiters et al 1998], the case base was used for offline purpose. That is, the case base was not used for actively participating in handling exceptions at runtime. It was used for gathering knowledge.

In the work of [Luo et al 2000], the 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 as 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 previous gained exception handling experiences in the future.

The work reported in [Klein and Dellarocas 1998] was more related to coordination science, such as high-level conflict management. This work was based on the Process Handbook project at the MIT Center for Coordination Science. It involved a decade of development and evaluation of systems for handling multi-agent conflicts in collaborative design. The exception handling discussion in this work was very high level and its applicability was not convincing, as their discussion is conducted without a workflow prototype system. A knowledge base where knowledge was acquired totally by humans was used in this work to store generic process templates captured. This knowledge base was used only for helping people learn to resolve conflicts. This work would be more valuable if generic exception handling expertise in their repository was available that can be used directly in a process management system.

In [Klein 2000], a semi-formal web-accessible repository of exception handling expertise was organized for the learning purpose. Their classification of exception or exception taxonomy was comparable to the exception categorization discussed in [Luo et al 1998, Luo et al 2000]. Their linkage to/from the exception type taxonomy was similar to the one proposed in [Luo et al 2000]. Their four main classes of exception handling processes were similar to the 3-D exception model in [Luo et al 2000].

CROSS-ORGANIZATIONAL EXCEPTION HANDLING

In [Ludwig 1999], an exception-handling mechanism was used to terminate business processes outsourced. Until now, there has been little progress in cross-organizational exception handling, though people have come to understand the importance of the research problems.

ADAPTIVE WORKFLOW

Organizational processes are often dynamic. They evolve over time and often involve uncertainty. Exceptions are usually unavoidable. To adapt to its environment and to resolve exceptions, WfMSs should be flexible enough that necessary modifications to workflow specifications and instances are allowed. They need to be complemented with execution support or run-time solutions such as dynamic scheduling, dynamic resource binding, runtime workflow specification, and infrastructure reconfiguration. Developing systems that are able to support dynamic and adaptable workflow processes stands out as one of the difficult new challenges in the future evolution of WfMSs [Kochut et al. 1999]. Such systems must be uniquely sensitive to a rapidly changing process execution triggered by collaborative decision points, context-sensitive information updates, and other external events. Some research issues in this area that have been raised in the context of modeling and specification aspects appear in [Han and Sheth 1998] and the relevant issues involving organizational changes appear in [Ellis et al. 95, Hermann 95]. The majority of current work addresses relevant issues at modeling and language levels [Krishnakumar and Sheth 95, Ellis et al. 95, Jablonski et al. 97, Han 1997], with few efforts on implementations underway [McClatchey et al. 1997, Taylor 1997, Reichert and Dadam 1998, Kochut et al. 1999]. A particularly different approach to supporting adaptive workflow (capable of reacting to the changes in local rules and other conditions) was investigated using the notion of migrating workflows [Cichocki et al. 1997].

In [Han et al 1998], the authors highlighted the needs for adaptive workflow management. Four levels of workflow adaptation were discussed - domain, process, resource, and infrastructure. Potential mechanisms for adaptive workflow management discussed are meta-model, open point, and synthesized.

In [Carlsen and Jorgensen 1998], an approach of process modeling targeting unstructured or partly structured workflows was use. It was based on a modeling language called PPM [Gulla, et al. 1991; Willumsen, et al. 1993].

In [Reichert and Dadam 1998], authors presented an approach to runtime changes of in-progress workflow instances. Through analysis of data flow and control flow, several modifications were proposed. The correctness of these modifications was also verified. However, this work did not answer the question how the already running workflow instances are managed when modifications are made.

In [Joeris and Herzog 1998], to cope with the co-existence of workflow instances following either the old or the new schema, mechanisms for the versioning of workflow schemes and instances were proposed.

In [Liu and Pu 1998], the author proposed a family of activity-split and activity-join operations to restructure ongoing activities. Similar to other approaches such as [Reichert and Dadam 1998], they proposed a workflow correctness criteria based on their Reference Activity Model. They discussed cases where the re-constructions were allowed and where they were illegal.

In [Casati et al 1998], the authors provided a concept model of workflow. They conducted workflow evolution according to activity graphs. Based on the observation that workflow specification may be modified during runtime, in which workflow evolution may imply losing all or some of the work done, a set of primitive and evolution policy are presented in this work. However, an important issue like how transactional properties of workflow execution are maintained is missing.

ADVANCED TRANSACTION MODELS

Workflow technology targets supporting reliable and scaleable process executions. In case of failures, workflow processes can resume their executions from one of their saved states, called a checkpoint, achieved by persistently saving the states from time to time. The activity of restoring a checkpoint and resuming the execution from the checkpoint is called rollback. The objective of failure recovery in workflow management is to enforce the consistency of the workflow under various failure scenarios [Rusinkiewicz and Sheth 1994]. The workflow should eventually reach an acceptable state after recovering from a failure in any of the workflow processing components. In real-world workflow applications we have seen that most tasks are non-transactional, and often involve long-lived tasks. They don not support the strict Atomicity, Consistency, Isolation, and Durability (ACID) properties of transactions. In the hope of using the complete ACID transaction theory, by relaxing these ACID properties, researchers have proposed many advanced transaction models.

NESTED TRANSACTIONS

A Nested transaction [Moss 1981] flags a milestone in the transaction model evolution. In this transaction model, a transaction may contain any number of sub-transactions. It extends the flat transaction structure to multi-level structures, usually called transaction trees. A child transaction may start after its parent has started and a parent transaction may terminate only after all its children have terminated. The nested transaction model decomposes a transaction hierarchically. In case of failure, the recovery can operate at the granularity of a sub-transaction.

OPEN NESTED TRANSACTIONS

Compared with nested transactions, open nested transactions [Weikum and Schek 1992] relax the isolation requirement in ACID properties. It makes the results of committed sub-transactions visible to other concurrently executing nested transactions with which they can commute. Two transactions can commute if each of their execution orders will result in the same final state.

SAGAS

The Saga [Garcia-Molina and Salem 1987] adds the capability to automatically determine and start compensation functions to the transaction environment. In its simplest version, a Saga is a sequence of transactions that either all commit, or compensation functions will be run for all already committed transactions. It relaxes the full isolation requirements and increase inter-transaction concurrency. Nested Sagas, an extension to Saga, relaxes atomicity whereby forward recovery is used in the form of compensating transactions to undo the effects of a failed transaction.

ConTracts

Reuter [Reuter 1989] first proposed the ConTract model. It combines ACID semantics with compensation. Its supported targeting applications are long-running computations. Long running process executions must be forward recoverable in ConTract. When a ConTract application is interrupted, the system must not redo steps that have already been performed successfully but it must resume the execution of the application where it left off because of the erroneous situation. Similar to Sagas, a ConTract is allowed to externalize its partial results before the whole ConTract is complete.

Stream flow

In stream flow [Leymann and Roller 2000], different parts of the workflow are assembled into a unit called work item stream to incorporate business-oriented units of work into transactional workflow. All the work item streams are assigned to a particular agent. All work items are created by the same workflow. All work items are the result of scheduling consecutive activities of the underlining process model. The same agent performs all work items of the stream. There are three patterns of work items in stream flow.

- Micro script stream. A micro script stream is a series of automatic activities whose implementations do not require user interactions. It is similar to the automatic task type in METEOR workflow model.
- Transaction stream. It includes a series of activities that are implemented by short-lived transactions. It is similar to the transactional task type in METEOR workflow model.

- Work package stream. It includes a series of activities that represents a whole complete work package for agents. In METEOR workflow model, the network task type is used for the same purpose.

FLEXIBLE TRANSACTIONS

Flexible transactions [Elmagarmid et al. 1990, Zhang et al. 1994] relax the isolation requirements. They allow transaction designer to specify acceptable termination state, specifying a set of functionally equivalent sub-transactions. Each of these sub-transactions when complete will accomplish the task. A flexible transaction is resilient to failures in the sense that it may proceed and commit even if some of their sub-transactions fail.

MULTI-LEVEL TRANSACTIONS

Multi-level transactions [Gray and Reuter 1993, Weikum and Schek 1992] are a specialization of open nested transactions where the tree of sub-transactions is balanced. Sub-transactions are allowed to commit and release their resources before their higher level transaction successfully completes and commits. If their higher level transaction aborts, those already committed sub-transactions may be undone by executing compensating sub-transactions. Other sub-transactions that observe this effect will be compensated too. One advantage of Multi-level transaction model is that less locking is required because operations at high level are allowed to commute even when operations at lower level might not be.

CROSS-ORGANIZATIONAL WORKFLOW

With the advent of Internet commerce, WfMSs have been increasingly deployed to deliver e-commerce transactions across-organizational boundaries. One basic issue is how to connect these WfMSs across organizational boundaries [Ludig et al. 1999]. In CrossFlow [CrossFlow] project, a virtual enterprise coordinator is used to manage the interactions among WfMSs. It is configured according to the business agreements that have been reached through contracting among participating organizations. Several other approaches have been proposed in [Ludig et al. 1999, IDSO 2000] to address the following two problems:

- How to provide means for the integration of processes of different organizations while maintaining each organization's privacy.
- How to manage the dynamics of the relationships between several organizations.

There are some other open problems [Ludig et al. 1999]. One problem is how to obtain a common view of a process being enacted in multiple organizations. Another problem is how to decide which organization can monitor and control the process performance in other organizations.

Business processes can operate within, across or between organizations in order to implement business value chains to deliver E-Commerce transactions. Two basic means to construct these cross-organization workflows are bottom-up and top-down. In the top-down approach, an overview picture of the whole workflow must be obtained before they can be constructed. In the CMI [Georgakopoulos et al 1999], a project at MCC, inter-organizational workflow was achieved by using a process model sitting on top of other workflow models to map methods and tools for defining processes that compose services provided by different companies. In the case that if different workflow segments are available, the whole workflow may be implemented using a set of these workflow definitions that have already been created to support discrete segments of the overall process. In WISE [Lazcano et al. 2000], a web based e-commerce platform project, process designers can post their design segments into a common World Wide Web (WWW) based catalog repository. A virtual process specification can be constructed by using the segments retrieved from the catalog.

This specification can then be compiled and the resulted processes will be enacted by the WISE engine. In RosettaNet [RosettaNet 2000], a Partner Information Process (PIP) is defined to allow business processes to interact with each other. An implementation framework is defined to form a standard for various concrete implementations. Similarly, IBM has defined tpaXML [tpaXML 2000], an XML based trading partner agreement specification language to facilitate communications between trading partners. This tpaXML has been integrated into ebXML [ebXML 2000], an e-business initiatives led by United Nations.

In summary, there are three ways we have seen of constructing cross-organizational business processes - split and deploy, composition, and black-box extension.

- Split and deploy. A whole process is designed. Then it is split into several sub-processes, and deployed in different organizations. This approach is currently supported by our METEOR project.
- Composition. In this approach, A whole process specification is obtained from several segments that may be contributed by different organizations. Then the process is built and deployed. WISE and CMI take this approach.
- Black-box approach. Contracting has been to interconnect different processes, forming certain agreements among cooperating processes. Several process segments are outsourced. A whole process specification, which might be incomplete, is usually available beforehand. CrossFlow takes this approach.

CHAPTER 3

CROSS-ORGANIZATIONAL BUSINESS PROCESSES

Organizations need to improve to survive. As organizations adapt to new conditions and respond to competitive pressures, business process re-engineering and process innovations that are key change management approaches have developed. These two approaches advocate a company wide approach to managing change. With the advent of Internet commerce, business processes involving business-to-business and business-to-customer activities usually span across multiple organizations. As a result, business processes become more dynamic. At the same time, they have reached almost anywhere where Internet can reach. To reach more customers and do business in a more economic way, organizations seek to partner with other organizations. Thus, a single organization no longer tries to own the whole business processes to achieve its business goals. Instead, organizations utilize the business processes of other organizations. They outsource their not-core business processes while keeping its core business based on the cost-benefit analysis. This has stimulated research interest in cross-organizational business process re-engineering and innovations.

In this chapter, we explore techniques that support cross-organizational business processes. This forms common background for discussions in later chapters. We present here an approach of process modeling. Then we elaborate the dynamics of the process chains. At the end of this chapter, we will discuss try-catch block exception handling mechanism and the exception handling inside a WfMS and across organizational boundaries.

PROCESS MODELING

Business processes are deployed in organizations. Each organization can decide that whether its processes are accessible by external parties and who are allowed to access to them. When external users access these exposed processes, utilization requirements are usually enforced to ensure the correct process assessment. They are usually specified in the process interaction requirement.

PROCESS OWNERSHIP

Business processes are owned by organizations. Organizations rely on these processes to achieve their business goals.

The processes deployed in organizations are viewed as value-added assets to the organizations. The ownership to the business processes by organizations enables them to have the right to control, monitor and outsource their owned processes.

PROCESS AUTONOMY

Process autonomy here refers to process monitoring and control besides process operation service accessibility. We use public, protected, and private to mark a process's autonomy. In the following, we explain process autonomy in the categories of accessibility, monitoring and control.

PROCESS ACCESSIBILITY

Accessibility of a process is whether the operation services provided by the process are available and/or accessible.

- If a service of a process is marked public, it is freely accessible by external processes or users.
- If a service of a process is marked protected, it is accessible by external processes or users under certain agreements. Process access contracts are necessary to ensure the correct access.

- If a service of a process is marked private, it is not accessible by external processes or users.

PROCESS MONITORING

Monitoring ability of a process is whether the process can be monitored. Here it refers to whether the monitoring services are available and/or accessible.

- If the monitoring service of a process is marked public, external processes or users can freely monitor this process.
- If the monitoring service of a process is marked protected external processes or users can monitor this process under certain agreements. Process monitoring contracts are necessary to ensure the correct monitoring access.
- If a process is marked private, external processes or users cannot monitor this process.

PROCESS CONTROL

Controlling ability of a process is whether the process can be controlled. Here it refers to whether the controlling services are available and/or accessible.

- If the control service of a process is marked public, this process is freely controllable by external processes or users.
- If the control service of a process is marked protected, this process is controllable by external processes or users under certain agreements. Process control contracts are necessary to ensure the correct access.
- If the control service of a process is marked private, this process is not controllable by external processes or users.

PROCESS INTERACTION

Processes interact with each other, thus forming a process web. There are several ways of process interactions, e.g., one way, or two-way interactions. These interactions are conducted through interaction points. Interaction points are the communication channels through which process information and/or control are exchanged. They are either two-way or one-way communication channels.

Process interaction points are the windows that processes expose their autonomy status to external processes and users.

Process interaction points are either waiting or non-waiting channels. Through a waiting interaction point, process information or control is exchanged immediately, while the interaction requestor will be waiting to get response back from the interacted process. Through a non-waiting interaction point, process interactions terminate immediately once the process information or control is exchanged.

PROCESS DYNAMICS

Here we discuss process dynamics in the areas of process inter-operability, process contract, and process fulfillment.

PROCESS INTER-OPERABILITY

Business processes operate within, across or between organizations in order to implement value chains. Because a set of workflow definitions have already been created and deployed in different organizations, to construct the overall process, one way is to inter-connect these already defined workflow segments. Process interoperability enables these different processes to talk to each other by exchanging messages. To achieve the process interoperability, several standards have been created.

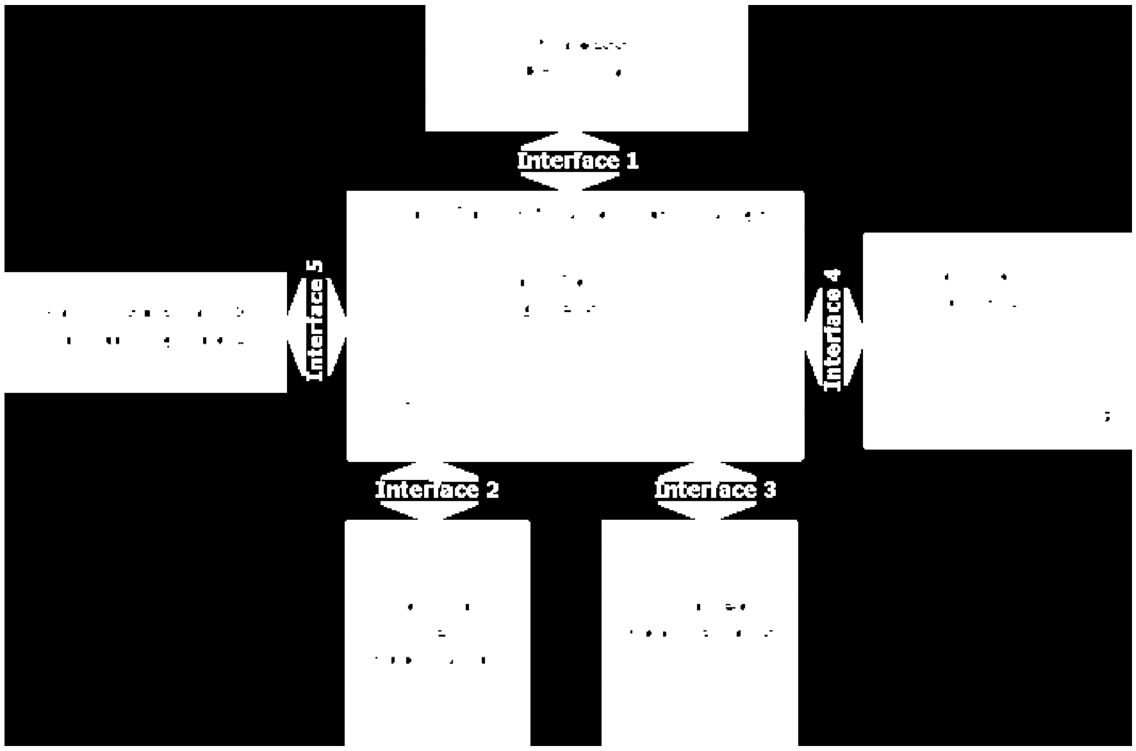


Figure 3.1 WfMC workflow reference model

In 1998, the Simple Workflow Access Protocol (SWAP) was created to provide a simple interoperability among Internet based WfMSs. It is a collective effort through an industry consortium under the auspices of many companies such as Netscape, Oracle, and Sun. Recently a specification called Wf-XML [WfMC XML] is created using an XML language designed to model the data transfer requirements set forth in the Workflow Management Coalition’s Interoperability abstract specification. It is an interoperability standard defined by a WfMC [WfMC] working group that combines the elementary concept of SWAP with the abstract commands defined by the WfMC interface 4 (See Figure 3.1). A similar interoperability standard is JoinFlow [JFLOW]. Its submission is also a joint effort by 19 companies on behalf of the WfMC. As an industry consortium the WfMC is not able to act formally as a submitter but fully endorses this submission as a supporter. The technology submitted is directly based upon the WfMC standards for workflow interfaces, which have been available in the public domain for a number of years and provide a base for the introduction of workflow technology into the Object Management Group (OMG) architecture. The workflow coalition has agreed the following inter-operability conformance levels that can be achieved:

- Chained sub-process interoperability (See Figure 3.2). In this type of interoperability, a workflow process may invoke another process but might not wait for the other process to finish. It can proceed with its own implementation irrespective of the results of the other workflow. Both the processes would terminate independent of each other.
- Nested sub-process interoperability (See Figure 3.3). In such an example the invoking workflow waits for the other workflow to complete before finishing or continuing with its work. In this case the activity becomes the Requester and it serve as a synchronization point in the interaction of both workflows.

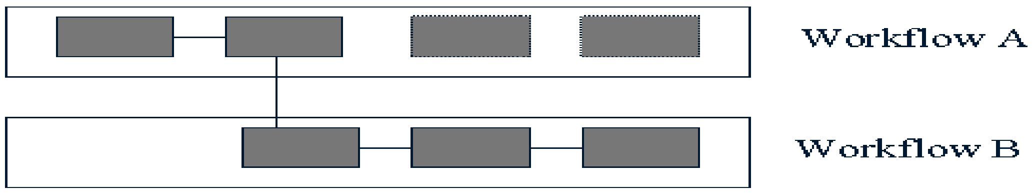


Figure 3.2 Chained sub-process inter-operability

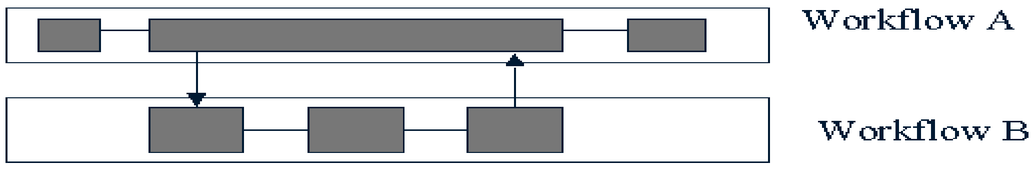


Figure 3.3 Nested sub-process interoperability

PROCESS CONTRACTING

Business processes are deployed to achieve business goals. Certain transaction environment must be established so the process interactions among business processes can be conducted. This environment is usually warranted via contracts. Processes interact with each other on the basis of these contracts. A contract usually consists the following items as discussed in some projects like CrossFlow [CrossFlow]:

- Process service identification
- Process partner identification
- Service semantics
- Control semantics
- Monitoring semantics
- Quality of service
- Security specification.

To reach a contact, the contracting parties must be identified. The subject of the contract must be clearly indicated. The time period of validity should be indicated clearly. An agreement of non-repudiation of the contract must be reached. The process of reaching agreements for a contract is actually the procedures for business process outsourcing [Halvey and Melby 2000]. This has also been discussed in various researches (e.g. CrossFlow [CrossFlow]):

- Information gathering. Contract concept may be formed at this stage. Organizations or processes search and gather necessary information to determine whether to use internal products or to outsource other organizations' processes.

- Contracting intention. The contract preparation activities may be conducted at this stage. In most cases, a request for proposal (RFP) is sent out to interested organizations or processes. The RFP contains the requirements for the contract to be reached.
- Agreement reaching. Contract negotiation activities are conducted in this stage. They are active until agreement is reached.
- Contract fulfillment. After agreement has been reached, the contract must be fulfilled in the business operations. In case of any exceptional situations, they must be resolved. Conflicts in the contract fulfillment must be settled.

The contracting activities among process supporting systems across organizations to form virtual enterprises vary according to the following two factors:

- Interoperability of process supporting systems.
- Autonomy of process partners.

According to the degree of the process interoperability and autonomy, the contract can be reached either statically or dynamically. If a contract is reached statically, a global process must be predefined before it is operational. Through dynamic contracting, global business process can be constructed on the fly.

CONTRACT FULFILLMENT

Once agreement has been reached, contract must be fulfilled when it is effective. There are several issues related to contract fulfillment: service delivery, contract fulfillment monitoring, and process control and exception handling.

SERVICE DELIVERY

To delivery the services, the quality of service (QoS) must be maintained. To guarantee the service QoS, process monitoring allows tracking the progress of outsourced services both online during service execution and off-line to analyze aggregated information. The end-users can be granted some control over the outsourced services (e.g., stop, start, and abort). Exception handling capability helps settle service fulfillment conflicts.

PROCESS MONITORING

Process monitoring is related to process autonomy. When a process exposes itself, external processes or users can monitor it. There are some issues in process monitoring:

- How much information is necessary to get a useful picture of the monitored processes?
- When such information is private, how can this information be secured and used as indicated in the contract?

PROCESS CONTROL

Process autonomy decides whether it is controllable by external processes or users. Some basic controls are start/stop and suspension/resume of processes. Other controls are creating new instances, deleting existing instances. More advanced controls are dynamic modifications of business processes. Due to their contradiction with process autonomy, dynamic process modification is usually conducted based on process reconfiguration.

EXCEPTION HANDLING

People have been used to exception handling as try-catch block as it is used in the language of C++ and Java. Try-catch method is useful for making programs less prone to crash. The basic mechanism of a try-catch exception handling is that when an exception occurs in the try block, if this exception or one of its super class is specified in the catch statement, then it will be handled by the code in the catch block. If this exception is not caught, it will be propagated according to the calling structure. When the exception is propagated to the operating system, this program is terminated.

Try-catch mechanism provides a structural programming means for programmers to write codes to handle errors. This structured try-catch mechanism also puts limitation on exception propagation. That is, an exception can only be propagated along the program calling sequence. One shortcoming of this approach is that when there is a better handler for this 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, its focus is on local exception handling which still fits the paradigm of structural programming. That is, though the exceptions in distributed systems are classified into local exceptions and remote exceptions; they are treated as the same and are handled locally when they are caught. So this try-catch mechanism really lacks the process-oriented view in handling exception handling in distributed systems, e.g., WfMSs. It puts challenges for programmers to flexibly propagate exceptions to capable handlers.

INTRA-WFMS EXCEPTION HANDLING

Try-catch exception handling is not suitable for handling exceptions in a WfMS. WfMS, a special type of distributed system, has its own exception handling requirements. In this section, we will focus on the exception handling in METEOR model 3 [METEOR Model 3]. We believe this discussion is general enough to cover various aspects in handling exceptions inside a WfMS.

In a WfMS it is assumed the all variables that describe the properties of processes are taken from a set of variables, called process variable set or vocabulary. Instances of those variables form the process environment. Situation of the activities in processes at a certain point of time is called a process state that is specified through task states and data states and the status of workflow environment. Inter-state dependence constraints are enforced through process transitions. A process is a series of process states linked by process transitions with starting and ending state specified. (Please refer to the section of process analysis in appendix).

In METEOR model 3, a *task* represents an abstraction of activity. A task can be regarded as a unit of work, which is performed by a variety of processing entities, depending on the nature of the task. A task can be performed by (realized) by a computer program, a database transaction, or possibly by a network of interconnected tasks called a *workflow*.

A task may be *invoked,* analogously to a procedure call. A task invocation creates a *task instance*, which, in case of a task being realized by a workflow, creates a workflow instance. A task instance *terminates* when its realization terminates. A task realization either succeeds or fails which is then reflected by the task entering its success final state (either *Done* or *Fail*) or its failure final state (either *Commit* and *Abort*). A task may enter its failure final state due to a failure of its realization, which is described by a suitable exception object. In this case, the task is said to *throw* this exception. Subsequently, such an exception may be used for scheduling of an alternate task in the workflow.

During the workflow execution, a number of undesirable events may occur. For example, one of the hosts used by a workflow application may crash, a network connection may go down, or possibly one of the tasks in the workflow application may detect an exceptional condition, specific to the task itself. In METEOR model 3, an undesirable event may fall into the category of a workflow management system-specific, or workflow application-specific. An exception that is system specific is called a system exception, while a workflow-specific exception is called an application exception. In METEOR model 3, system exceptions may occur during the execution of every workflow, while application exceptions are restricted to specific applications.

In METEOR model 3, an exception is viewed as an occurrence of some abnormal event that the underlying workflow management system can detect and react to. Any abnormal event that is not detected by the enactment system will not be considered to be an exception in METEOR model 3. This is from the implementation or enactment point of view. From the modeling point of view, this abnormal event is still an exception. We will discuss this again in the 3-D exception model in the chapter 3 of "exceptions in cross-organizational settings". Instead, in METEOR model 3, this abnormal event will be considered as external exception signal. 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 enactment and/or the workflow application may be notified externally of such an event by receiving an exception signal. An exception signal may be sent by an external program (for example, a database trigger), or manually, by a workflow administrator.

An exception handler is a description of action(s) that the workflow enactment system, or possibly a workflow application, is going to perform in order to respond the exception.

In order to provide an exception handler, we must consider:

- Location of control at the time of exception, and
- Exception type.

Placement of control forms a hierarchy of processing components according to the control structure in METEOR model 3. This hierarchy may be extended even further, if a workflow application has a hierarchy of tasks with network realizations (the actual task implementations). An exception may occur at any time, while control is within one of the components in the hierarchy (see Figure 3.4).

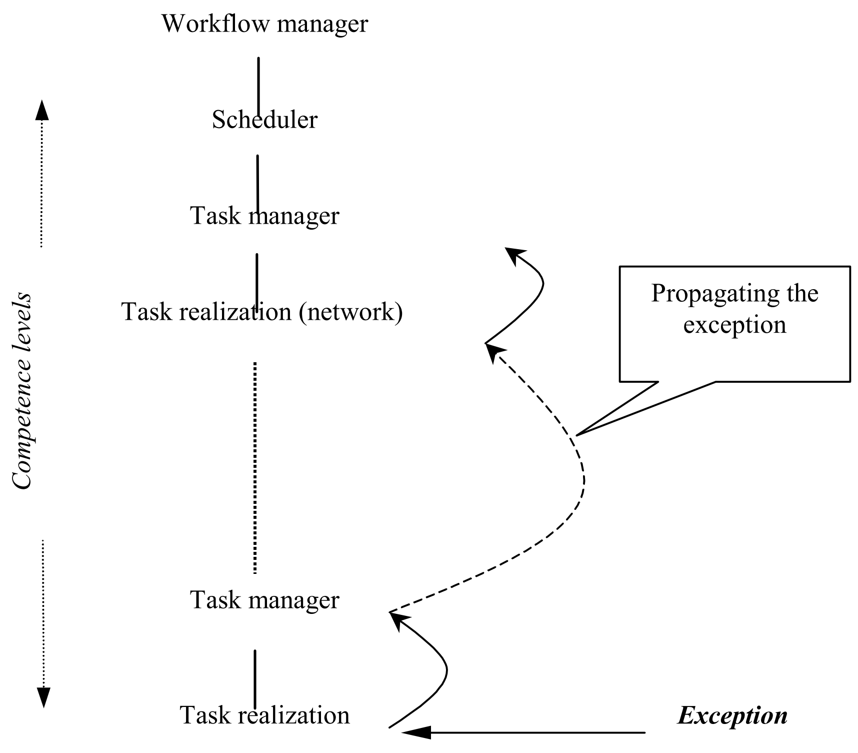


Figure 3.4 Competence hierarchy

A competence-driven exception handling is adopted in METEOR model 3 (see Figure 3.4). 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 resides. If the component is competent to handle the exception, that is it has at least one handler for the given exception type, it handles the exception. On the other hand, if the component does not have the competence for the exception, the exception is passed to the next higher level in the competence hierarchy.

If a component has been designed to handle a given exception, it does so using its own handlers. A component may also decide to send the exception to the higher level in the component hierarchy. If a component has not been designed for handling of the exception, the exception is passed onto the higher level by default.

To be more specific, if an exception occurs during an invocation of a task realization, its parent task enters its fail state and a suitable exception object is made available for scheduling purposes. This is analogous to the exception being thrown by the task.

If the task is part of a network, which is a realization of a higher-level task, the network may be competent to handle the thrown exception. That is, there may be a failure transition set for the given exception. Such a transition is called a handler for the exception. However, if the parent network is not competent to handle the thrown exception, the whole parent network fails and re-throws the same exception to its own parent, which may have the competence necessary for handling the given exception.

Since the exceptions form a hierarchy, a given task may have more than one suitable handler (for example, for a specific exception and for one of its predecessors in the hierarchy). In this case, the most specific handler is used.

It is possible that an abnormal event cannot be detected by any of the components of the workflow system. For example, a currently running workflow instance is in violation of a newly introduced business policy. It might be the case that the workflow runtime should force a workflow instance to fail the next task and continue its execution following one of its alternate paths. However, at this time neither the workflow application nor the workflow runtime system is aware of the abnormal event. Such an abnormal event is called an external fault.

An exception signal may be sent to the workflow system in order to make an external fault known to the workflow. The signal may be sent by an external entity, such as a workflow administrator, or a separate program. The workflow administrator may decide to force a particular workflow instance to fail one of its currently running tasks, and send a suitable exception signal to the task manager. Similarly, a process monitoring database updates may trigger sending an exception signal to the workflow manager. Another example of an exception signal may be a TimeElapsed signal sent to a task manager by the workflow runtime timer. The task manager may then issue a TimeElapseException as the result of a task not completed within a specified time.

When the exception is propagated to the workflow manager (see Figure 3.4), it would be propagated to an entity outside of the current workflow management system. Then this exception will be handled at the cross-organizational process level.

CROSS-ORGANIZATIONAL EXCEPTION HANDLING

Exception handling research in enterprise-wide workflows has achieved numerous results. For example, the exception handling strategy in METEOR model 3 is a much better approach than the basic try-catch mechanism for handling exceptions inside a WfMS. However, it primarily deals with intra-WfMS exception handling thus lacks the process oriented view for handling exception. 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 model 3 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 [Luo et al 2000] inside a single WfMS. When the exception is propagated to the workflow manager (see Figure 3.4), it would 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 propagated above workflow manager. We will discuss this more in the next chapter. 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.

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 chapters.

In the next chapter, we further discuss exceptions in cross-organizational business processes.

CHAPTER 4

EXCEPTIONS IN CROSS-ORGANIZATIONAL SETTINGS

In this chapter, we will first introduce our 3-D exception model. We then give a brief description of our enterprise-wide exception handling methods. After identifying the exception handling problems in cross-organizational settings, we briefly introduce our solution to these problems, i.e., a bundled exception handling approach that supports (1) exception handling knowledge sharing, (2) coordinated exception handling, and (3) intelligent problem solving.

3-D EXCEPTION MODEL

Exceptions in our view refer to facts or situations that are not modeled by the information systems or deviations between what we plan and what actually happen. Exceptions are raised to signal errors, faults, failures, and other deviations. They depend on what we want and what we can achieve. For example, in the infant transport application, space provided by an ambulance is limited. So is the transport time. The exception handling mechanisms might be different from that used in NICUs because of the differences in time, space and places. In most realistic situations and non-trivial systems, there are always interest conflicts between what we want and what we can achieve. It is more acceptable to design a system that can operate as best as it can; when there is an exception, it can be handled by the system. We call such a system an exception-aware system.

Exceptions provide great opportunities for the systems to learn, correct themselves, and evolve. To build an exception aware system, it is beneficial to clarify the nature of exceptions to get guidelines in the systems development. As shown in

Figure 4.1, known, detectable, and resolvable form three dimensions for the exception knowledge space. The *known* dimension is usually captured through exception specification. Supervision is one of the approaches to enlarge exception knowledge space in the *detectable* dimension. To resolve exceptions, capable exception handlers should be available that make up the *resolvable* dimension of the exception knowledge space. Any position in this exception knowledge space can be represented as an exception point. The exception knowledge of an exception aware system is the set of all these points.

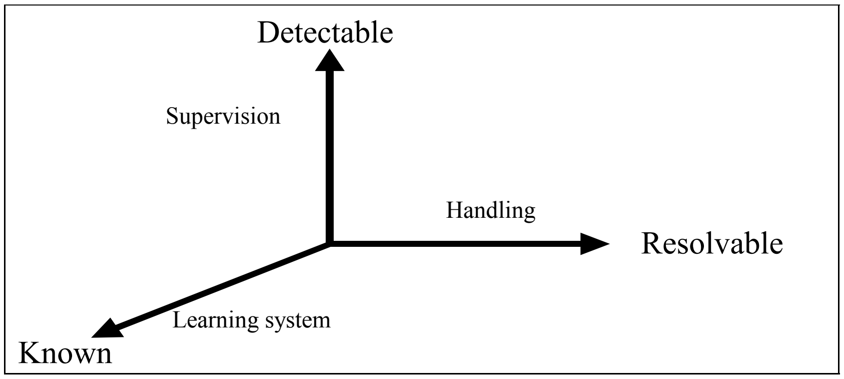


Figure 4.1 Three-dimensional analyses of exceptions

- **Known**: Every person's knowledge is limited. The same is true for a system. The world is governed by rules that either we know or we are still investigating, and our knowledge continues to expand through learning. As this learning process continues and unknown or uncertainties become known or certain, prior decisions made may be revised and other uncertainties may be considered. When a system cannot meet the new situation, exceptions occur. We call these kinds of exceptions unknown exceptions, since they are beyond the system's current knowledge. Otherwise, we consider them known. For example, during the transport of the newborn infant, an unknown exception may be caused by an abnormal situation that has not been met before. A basic solution to unknown exceptions is to build an open system that is able to learn and can adapte to handle those exceptions.

- **Detectable**: Exceptions can be classified based on a system's capabilities to detect an exception. If systems can notice the occurrence of an exception then we call it a detectable exception; otherwise we consider it undetectable. An unknown exception is usually undetectable because there is no way for the system to know about it until further improvement to the system is accomplished to achieve that capability. Sometimes an unknown exception might be detected as another known exception. Detection of an exception depends on the system's capabilities. For example, if there is no equipment on board to measure certain situations, e.g., neurologic checking of the newborn infant, exceptions related to neurologic situations might not be detected. Moreover, if there is no mapping from the errors occurring in the measuring equipment in which an equipment related exception should be raised to a workflow system's exception, that equipment exception will not be detected either. If the system can notice that there is an exception, it may be possible for the system to derive exception-handling schemes to handle such exceptional situations.
- **Resolvable**: Undetectable exceptions are not resolvable at the time of occurrence, for their occurrences are unknown to the system. Also, there are certain known exceptions that may be ignored during system modeling time due to certain specific reasons, such as the frequency of their occurrences is so low, the effect caused by the exceptions to the system can be ignored. When such exceptions actually occur, the system cannot handle them (e.g., the Y2K bug). For example, in the infant transportation application, on board there are necessary medicines for commonly

occurring health conditions for newborn infants. When a medicine is not on board but is needed for a situation that rarely occurs, then the corresponding exception to the process is not resolvable at that time. Based on the system's handling capability, exceptions can be categorized into two categories: resolvable and irresolvable. When exceptions occur, the system can derive a solution to resolve the deviations. Such exceptions are called resolvable exceptions. When an exception occurs, but the system cannot derive a solution to solve the deviation to meet the requirement, then it is called irresolvable exception.

Based on the above perspective, exception aware systems should be built to have adequate initial exception knowledge represented as exception points. To deal with exceptional situations, a system actually finds an exception point in the exception knowledge space through propagation and masking. Propagation allows a system to propagate an exception to a more appropriate system component to find an appropriate exception point. Masking usually means when there are several exception points, the one that is close to where exception is detected is the best candidate.

EXCEPTION HANDLING METHODS

Exception sources include errors, failures, and rule changes, etc. Possible exception handler candidates are retry, recovery, compensation, dynamic changes, etc. In this section, we briefly describe these exception methods. To use them in the cross-organizational settings, there are three items to be considered, i.e., knowledge sharing, and coordination of the exception handling process, and searching for appropriate handling method.

Workflow technology targets supporting reliable and scaleable execution of business processes involving both humans and legacy systems, in distributed and often heterogeneous environments. In case of failures, workflow processes usually need to resume their executions from one of their saved states, called a checkpoint, achieved by saving the states from time to time persistently. The activity of restoring a checkpoint and resuming the execution from the checkpoint is called rollback. Those techniques have long been used in database systems. A checkpoint is an action consistent checkpoint if it represents a state between complete update operations. A consistent state in the database domain is a state when no update transactions were active. This checkpoint representing a consistent state is a transaction consistent checkpoint. A checkpoint does not need to satisfy any consistency constraints. Such a checkpoint is often referred to as fuzzy. But recovery after failure must always guarantee that the resultant state is transaction consistent even though any checkpoint used may not be.

The objective of failure recovery in workflow management is to enforce the consistency of the workflow under various failure scenarios [Rusinkiewicz and Sheth 1994]. The workflow should eventually reach an acceptable state after recovery is from a failure in any of the workflow processing components. In real-world workflow applications we have seen that most tasks are non-transactional, and often involve long-lived tasks, thereby not supporting the strict Atomicity, Consistency, Isolation, Durability (ACID) properties of transactions. Hence, although desirable, it might not be possible to recover failed non-transactional tasks using backward recovery. The use of backward recovery for most human-oriented tasks is not a viable solution since most erroneous actions once performed cannot be undone. It might be possible for the human to rectify all the inconsistencies caused due to errors and redo the actions without affecting other tasks or data objects within the workflow. However, it would be rare to expect this behavior for most real-world human-tasks. Backward recovery is useful for purely data-oriented tasks that are transactional tasks or networks. Besides we also need a forward recovery mechanism that would semantically undo, or compensate a partially failed task. In order to recover the execution-environment context, appropriate status information must be logged on stable storage. Thus the state information can be restored at the time of failure, which includes the information about the execution states of each task and the scheduling dependencies. When failures occur, in most cases, roll back is necessary. In general workflow management system (WfMS) is not responsible for a task's internal rollback process. In a WfMS, tasks are treated as black box and they are expected to be responsible by themselves for a correct recovery.

In WfMSs, cooperating workflow processes are dependent on one another due to inter-process communication. To resume the workflow execution from failures, a set of consistent checkpoints, called a recovery line in [Park and Kim 1992], taken for such a group of inter-dependent processes, should be found. The recovery line can be maintained by properly coordinating each checkpointing and rollback activity; or an uncoordinated checkpointing approach can be used. In an uncoordinated approach, checkpoints can be independently taken, and during each recovery activity, a recovery line can be found by exchanging dependency information [Park and Kim 1992].

It might be useful to introduce the sphere of transaction into the workflow design and execution to calculate checkpoint and for recovery. The rational behind this is that usually the whole workflow cannot be transactional. Workflow systems need to coordinate both human tasks and automatic tasks. Usually a task that involves human decision making process, i.e. human task is not a transactional task. The recovery of a workflow usually will not undo all the work has been performed since it is too expensive and not practical. To support transactional recovery, it is necessary for the WfMS to support a flexible transaction model that can relieve certain ACID properties. This can be accomplished through a mapping process from this transaction model to workflow processes.

During the design of workflow, the sphere of transaction can be decided. Checkpoints can be calculated according to the spheres of transactions. The logging processes can be better scheduled according to these checkpoints. Necessary exceptions that should be checked will be defined at this time. During the workflow execution, runtime checking along with those predefined exceptions will be used to detect abnormal situations. However, when such exceptions occur, necessary actions will be taken.

PROBLEMS AND PROPOSED SOLUTION

The business environment is dynamic and hard to predict, so exceptions to business processes are unavoidable. It will be more challenging when these exceptions need to be routed among all the contracted business processes across organizational boundaries. For example, if a customer's connection requests were rejected frequently, the direct complaints should go to their SPs. However, the real problem may lie in L3, the telecommunication infrastructure provider. But L3 may not even be aware of the abnormal situation, since no exceptional events are routed to it. This has resulted in the strong need for a new exception resolution mechanism in cross-organizational settings. The fact that there exist different characteristics in such exceptions in cross-organizational processes from those in enterprise-wide processes poses special problems for cross-organizational exception resolution. These problems are summarized in Table 1.1. They are classified into the following categories:

- Heterogeneity: Exception specification may be different in various processes deployed in different organizations. Different organizations may have different exception handling policies. When deriving the exception resolutions, these heterogeneous exception sources should be taken into consideration and be integrated.

- Responsibility determination (Coordination): When an exception occurs, it or its resolution must be routed to responsible parties. Such exception routing across organizational boundaries needs special considerations. It needs coordination.
- Lack of understanding of the outsourced processes. Exceptional problem solvers are usually not quite familiar with the details of the outsourced processes. Exception handling experience sharing in cross-organizational settings seems more valuable than that in enterprise-wide processes in this context. It enables exception handlers to derive exception solutions in a more informed and efficient way.

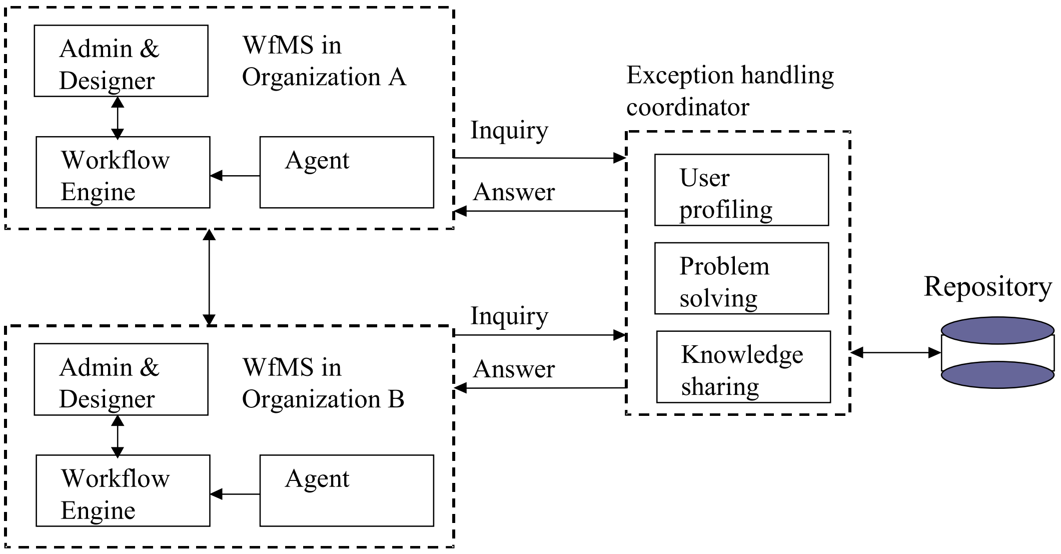


Figure 4.2 the conceptual model of the coordinated exception-handling scheme

With the advent of Internet commerce, it is increasingly common to see process outsourcing and dynamic business processes spanning organizational boundaries. Sophisticated exception handling mechanism in such an environment is becoming even more important. To support exception handling in cross-organizational settings, we have proposed a bundled solution (see Figure 4.2) to solve the problems, i.e., heterogeneity, responsibility determination (coordination), and lack of understanding of other party's business processes. This solution involves the following:

- 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 the case repository (see Figure 4.2).
- 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 4.2) coordinates the exception handling process. Five modes of coordinating exception handling in cross-organizational settings are proposed - immediate, deferred, de-coupled, free and close. They are based on the identified types of process interactions in cross-organizational settings to meet various business requirements.
- Intelligent problem solving. Exceptional situations are usually very complicated. A knowledge-based approach is a good candidate in dealing with such complicated situations. It helps workflow designers and participants better manage the 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. It allows the users to navigate through the knowledge repository to find support for his/her decision as to how to handle a certain exception. An explanatory module is incorporated

into the knowledge systems to explain the exceptional situation and solutions.

CHAPTER 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 to make decisions. 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 [Bittel 1964, Harvard 1998]. 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. Exception handlers include retry, compensation, alternative task, and recovery, etc.
- Exception handling experience, which will be acquired in case representations.

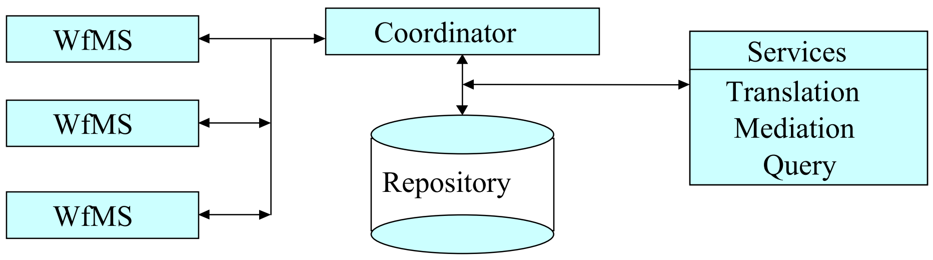


Figure 5.1 Exception knowledge sharing

These patterns, described using a set of attributes, are stored in the case repository (see Figure 5.1). They are shared by all cooperating business processes.

The case repository is built upon the workflow repository (See Figure 5.2). Detailed information about workflow repository can be found in [LSDIS 2000]. A set of interfaces should be provided to query the workflow repository to allow the knowledge sharing. These interfaces support querying the following:

- the structural information about workflow specifications
- exceptions that occur in the cooperating organizations
- exception handlers for specific exceptions
- responsible party for the abnormal situations
- interaction points among cooperating business processes

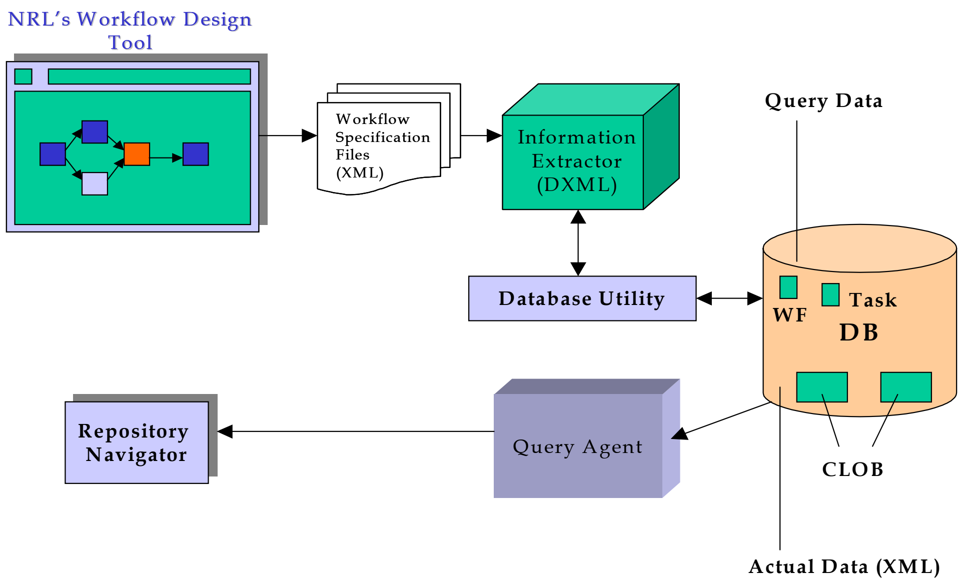

NRL's Workflow Design Tool
Workflow Specification Files (XML)
Information Extractor (DXML)
Query Data
Database Utility
WF
Task
DB
Repository Navigator
Query Agent
CLOB
Actual Data (XML)

Figure 5.2 Workflow repository architecture

EXCEPTION SPECIFICATION

Exceptions can be divided according to organizational boundaries. One type of exceptions is enterprise-wide exception. The other type is cross-organizational exception. Three broad exception categories can be made upon these enterprise-wide exceptions: infrastructure exception, workflow exception, and application exception

[Luo et al. 1998, Luo et al. 2000]. 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 system 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, 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 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. A runtime exception within a task that is caused due to memory leaks is another example of application exception. 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 performing the change assignment.

In the cross-organizational settings, another exception category cross-organizational exception should be defined. A cross-organizational exception is an infrastructure exception, a workflow exception, or an application exception. If it occurs it will affect the outsourcing fulfillment, or it may not be handled alone in one outsourcing partner. An example of such an exception in the telecommunication application is an application exception, which is caused by an error in the bandwidth change request that the agents couldn’t be found for the roles required to perform the change assignment. Because no agents can be found to perform the requested customer services, customers of the SPs are denied services, which directly harms the SPs' images.

EXCEPTION HANDLERS

Workflow systems need exception detection and handling mechanisms to satisfy reliability requirements [Luo et al 1999]. The exception-handling construct, as well as its supporting mechanism, is meant to be sufficiently general to cover various aspects of exception handling in a uniform way. It can help separate the modules to handle unusual situations from the modules for the normal cases. In fact, an exception is considered a learning opportunity for the workflow systems. It signals the abnormal situations of workflows and workflow execution.

In [Luo et al. 1998, Luo et al. 2000], various exception handlers are identified in workflow systems. These handlers work in a traditional workflow management system.

- Ignore. An exception is ignored, if no actions are taken to handle it.
- Record. An exception will be recorded when it occurs. The storage place can be a log file or a database.
- Retry. Only repeatable action can be retried. An example is database connection. 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 taken 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 workflow state from which the workflow execution can resume and proceed. This state may not necessarily satisfy the global correctness criteria.
- Propagation. If no local handler is available, then the solution is to route it to another workflow component, say exception-handling coordinator which is aware of more handling schemes.
- Termination, suspension, and resumption of processes. They are the basic functionality of 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 in 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 likely will fail. This request can be retried after waiting for a period of time that the server has started. In most cases, a simple retry will not solve the problems encountered. Compensation and alternative task are possible candidates. Recovery based solutions usually work only in transactional environment. Termination, suspension, and resumption of workflow processes are basic supporting functionality for workflow exception handling. Propagation and procedural exception handlers are too generic unless good templates are available.

In cross-organizational settings, we believe compensation preceding rework (CPR) is a good exception-handling template. Here, compensation means semantic compensation that is rationale based. Compensation is reached through certain communication efforts when both parties agree. If the effects of an operation can be totally undone, it is called perfect compensation. If no action can be taken to decrease the effects, it is called no compensation. Otherwise, it is called partial compensation. We will discuss CPR again in the section of coordinated exception handling. The CPR is contained in the action information block in the case data structure. It includes two sub-blocks, compensation block and rework block. The following is 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.

EXCEPTION HANDLING CASE

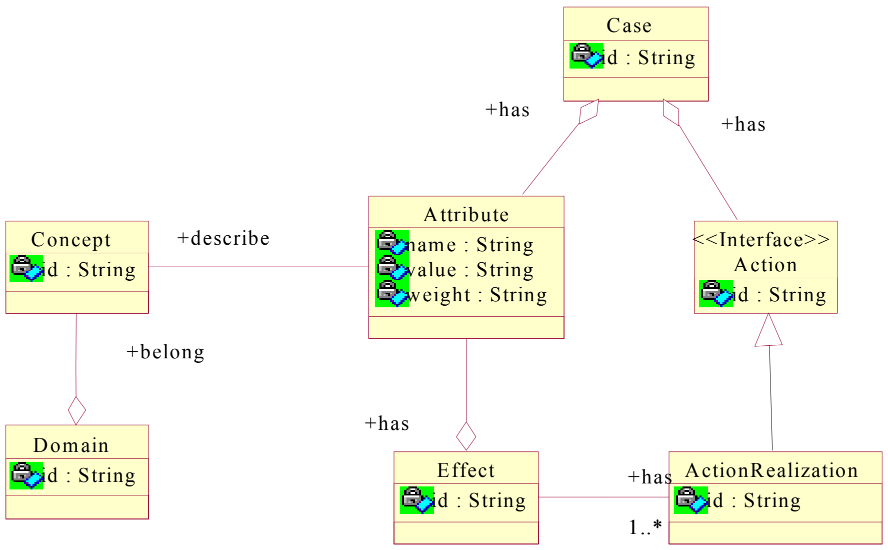


Figure 5.3 A conceptual model for case

We propose here a case based experience knowledge representation (see Figure 5.3). We are going to model the exception handling experience into cases. The patterns of the handling process will be derived by using a case-based reasoning (CBR) scheme, or through expert survey. Cases are data structures that are used to represent knowledge about exception handling experience. A case consists of descriptions of exception handling knowledge as well as exceptional situations. An exception case has a list of attributes specified in the format of <name, value [, weight]>. Name holds the type of the attribute. An attribute has value. Weight signifies the importance of this attribute in the case. These attributes are 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. Value of the attributes can be either constant, default, or free. An attribute with constant value cannot change. A default value gives expected value for that attribute. The value of an attribute can be modified if it accepts free values.

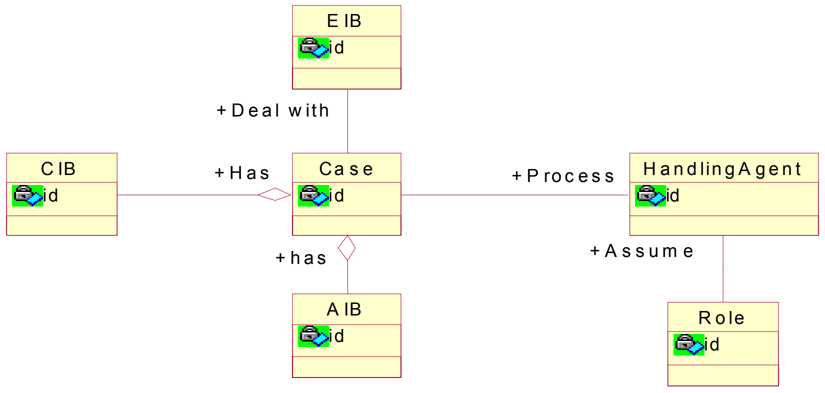


Figure 5.4 Relationship between case, exception information block and handling agent

A case (see Figure 5.4) contains three information blocks, EIB, CIB, and AIB. An exception information block (EIB) describes an exception. A context information block (CIB) records context information about the exceptional situation. An action information block (AIB) is used to record the actions taken to handle the exception situation. It is actual the CPR exception handling template. It contains two blocks - compensation block and rework block. These two blocks are used to describe the exception handling schemes. An example of such a case is as follows (See Figure 5.4). 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 block) actually says to insert the task again into the implementation repository and retry the task.

<Exception information block>

- Exception type, *Java null pointer*
- Exception message, *Meteor.OrbWork.OWTaskException: JAVA null pointer*
- Workflow name, *infant*
- Workflow ID, *infant0*
- Component name, *allocate_resource*
- Propagator, *reserved*
- State, *fail*
- Originator, *reserved*
- Key, ---
- Data, *r@*
- Host name, *mitchell.cs.uga.edu*

<Context information block>

- case_number, infant_routing, string, *2000-10-01-001*
- infant_name, infant_routing, string, *John Calton*
- infant_age, infant_routing, string, *4 days*
- infant_contact, infant_routing, string, *Smith Calton*

- infant_weight, infant_routing, string, *9*
- infant_situation, infant_routing, string, *weak*
- infant_diagno, infant_routing, string, *fine*
- infant_history, infant_routing, string, *ok*
- hospital, infant_routing, string, *Athens Best*
- hospital_contact, infant_routing, string, *Peter Duncan*

<Action information block>

- Compensation mode, *automatic*
- Compensation type, *Meteor.OrbWork.ExceptionHandler*
- Conpensation action, *register_server*
- Compensation parameter number, 2
- Workflow, *infant*
- Task, *sender*
- Rework mode, *automatic*
- Rework type, *retry*
- Rework task name, *sender*
- Rework state string, *START@start*
- Rework parameter string, *r@start@start*
- Rework host, *mitchell.cs.uga.edu*
- CPR mode, *user*
- CRP name, *cpr_ehc_wfi*

CHAPTER 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 [Luo et al 2000]. To solve the problem of 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, the WfMSs will report cross-organizational exceptions to the exception handling coordinator. The coordinator, then, will share this information with co-operating business processes.

COORDINATION MODE

Processes are deployed in different organizations. Organizations, as the process owner, can decide to their own benefit as how to and what to expose their processes to the outside. In the section, we will propose an approach of exception aware interconnections to provide guidelines to help resolve the conflict between operation flexibility and process privacy.

In the following, we will propose five modes of coordinated exception handling for handling cross-organizational exceptions. They are proposed to meet various business operation semantic requirements. These five exception-handling modes are built upon possible process interactions. Generally there are two basic types of process interactions: one-way interaction and two-way interaction. More complicated interactions can be represented by using these two basic types of interactions. In a one-way interaction, one process can send a message (request) to another process. After that, no interaction exists between the two processes. In a two-way interaction, one process can send a message (request) to another process. It can either receive response immediately or at a later time.

IMMEDIATE MODE

Immediate cross-organizational exception handling mode is used when there is a single two-way interaction point between processes deployed in different organizations. The exception information is exchanged through this single interaction point. For example, as shown in the Figure 6.1, SPs route their bandwidth change requests to L3 due to a growth in customer needs. They will immediately get an exception in case L3 determines that requested bandwidth range does not exist. L3 can register the occurring exception with the Exception Handling Coordinator (EHC).

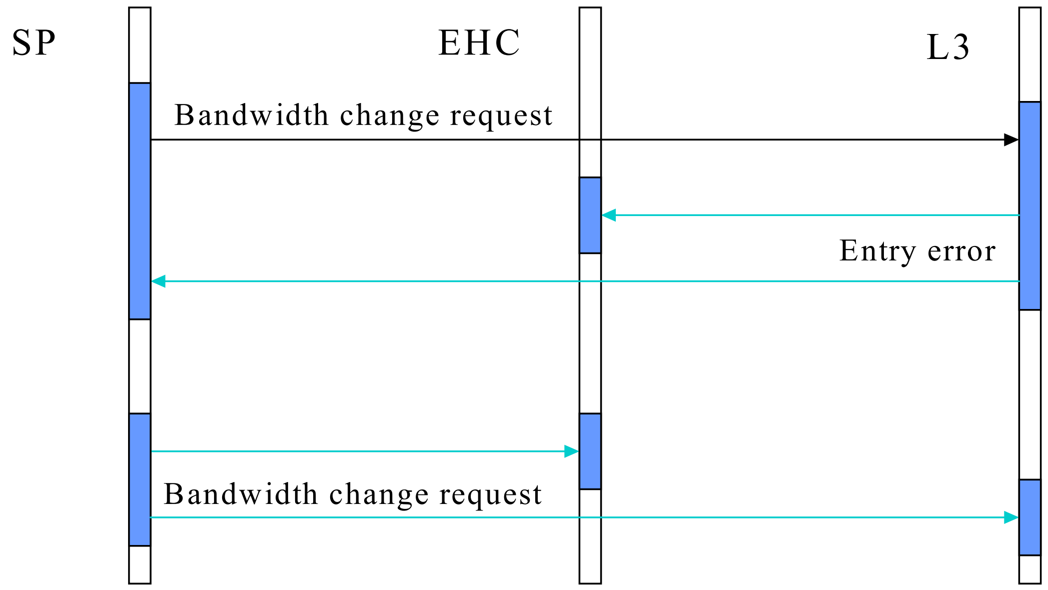


Figure 6.1 Immediate mode of cross-organizational exception handling

In the immediate handling mode, the CPR handler template is as follows.

- If (mode is immediate) signal the exception directly;

- If (compensation is auto/manual) execute the compensation scheme;
- If (rework is auto/manual) execute the rework scheme;

DEFERRED MODE

Deferred cross-organizational exception handling mode is used when there is a single two-way interaction point between processes deployed in different organizations. The difference from the immediate handling mode 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 6.2, customers send their subscription requests to SP. SP will not raise exception immediately to its customers when it determines that not enough credit information is available along with the request. Instead, SP will fulfill the request, and later raise the exception to customers through the same two-way interaction point.

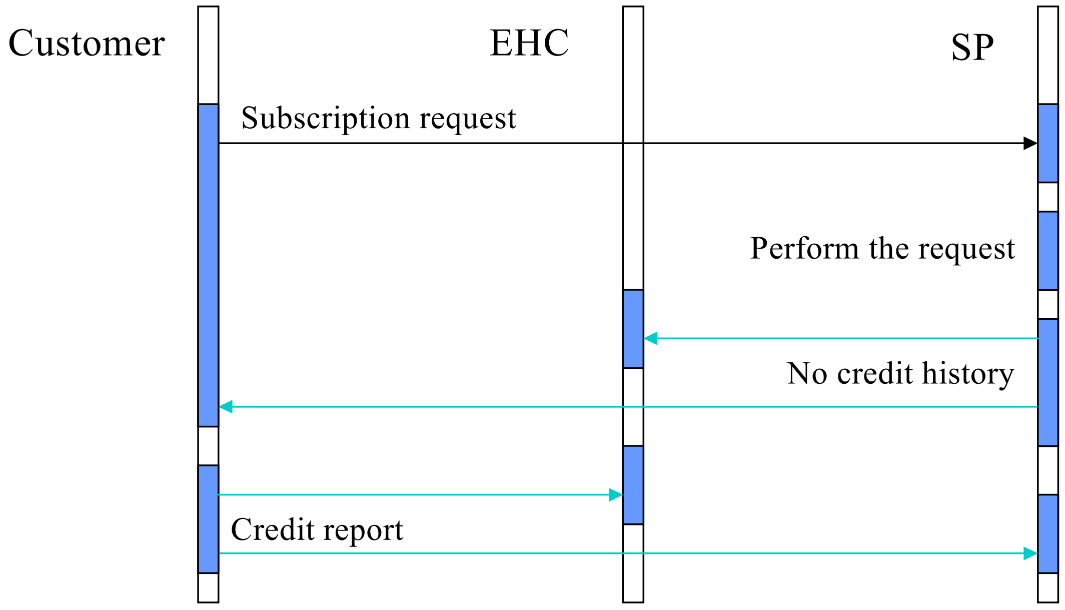


Figure 6.2 Deferred mode of cross-organizational exception handling

In the deferred handling mode, the CPR handler template is as follows.

- If (mode is deferred) fulfill the request and add the exception object to the normal flow;

- Signal the exception;
- If (compensation is necessary) execute the compensation scheme;
- If (rework is necessary) execute the rework scheme;

DE-COUPLED MODE

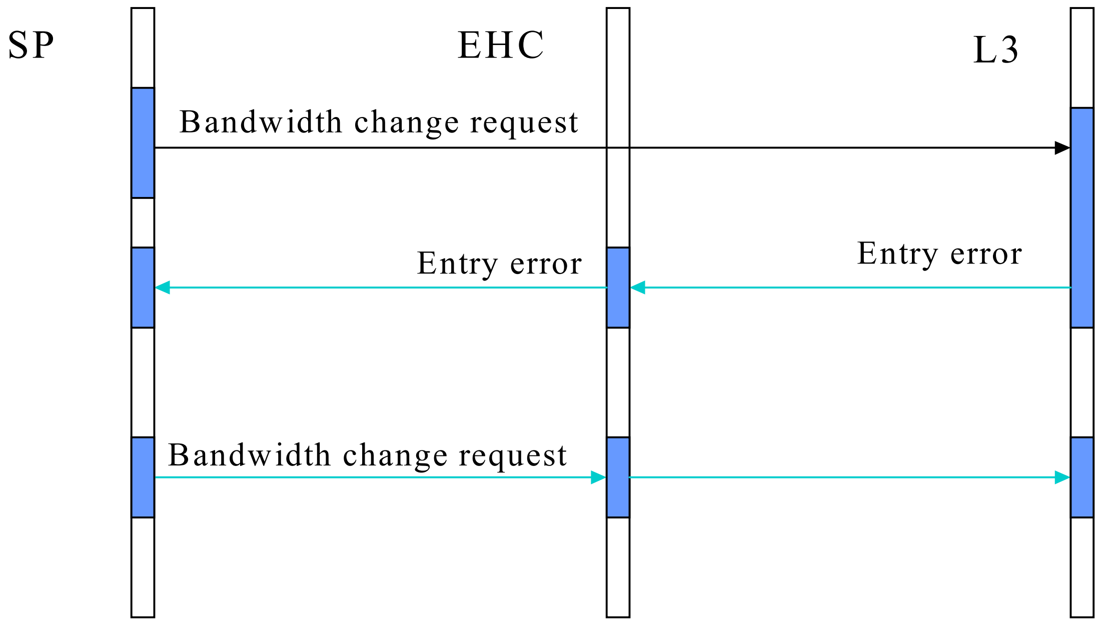


Figure 6.3 De-coupled mode of cross-organizational exception handling

De-couple cross-organizational exception handling mode is used when there is only a single one way interaction point between processes deployed in different organizations. For example, as shown in the Figure 6.3, 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 514KBPS channel is not available, it needs to raise this exception to SPs. Since there is no other interaction points between them, L3 needs to raise it to SPs through the exception handling coordinator.

In the de-coupled handling mode, the CPR handler template is as follows.

- If (mode is de-coupled) find the contact party in cooperating process and the interaction point;
- Signal the exception through exception handling coordinator;

- If (compensation is necessary) execute the compensation scheme;
- If (rework is necessary) execute the rework scheme;

FREE MODE

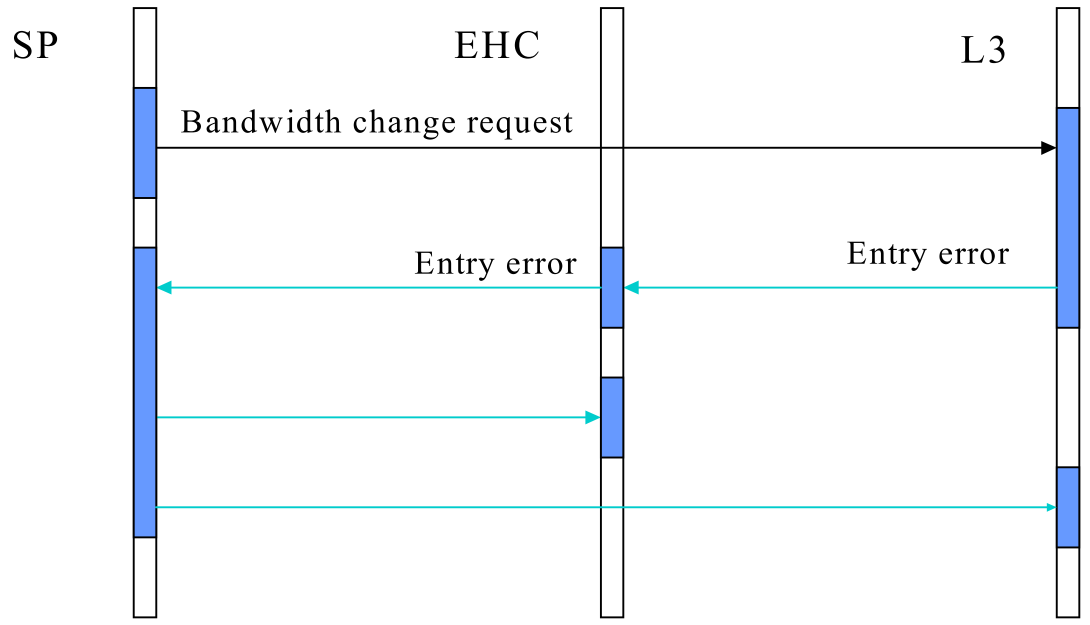


Figure 6.4 Free mode of cross-organizational exception handling

Free mode cross-organizational exception handling is used 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 6.4, SPs route its customers' bandwidth requests to L3. L3 will not raise an exception to SPs if it determines that an unrealistic number of channels are typed in along with the request. Instead, L3 will fulfill the request partially, and add an exception object to its normal flow. Later it will raise the exception to SPs through another interaction point determined by the exception handling coordinator.

In the free handling mode, a CPR handler template is as follows.

- If (mode is free) find the contact party in the cooperating processes and the interaction point;
- Add an exception object to the normal flow;

- Signal the exception;
- If (compensation is necessary) execute the compensation scheme;
- If (rework is necessary) execute the rework scheme;

CLOSE MODE

Close mode 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 the Figure 6.5, SPs determines the service quality provided by L3 is not satisfactory, since 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.

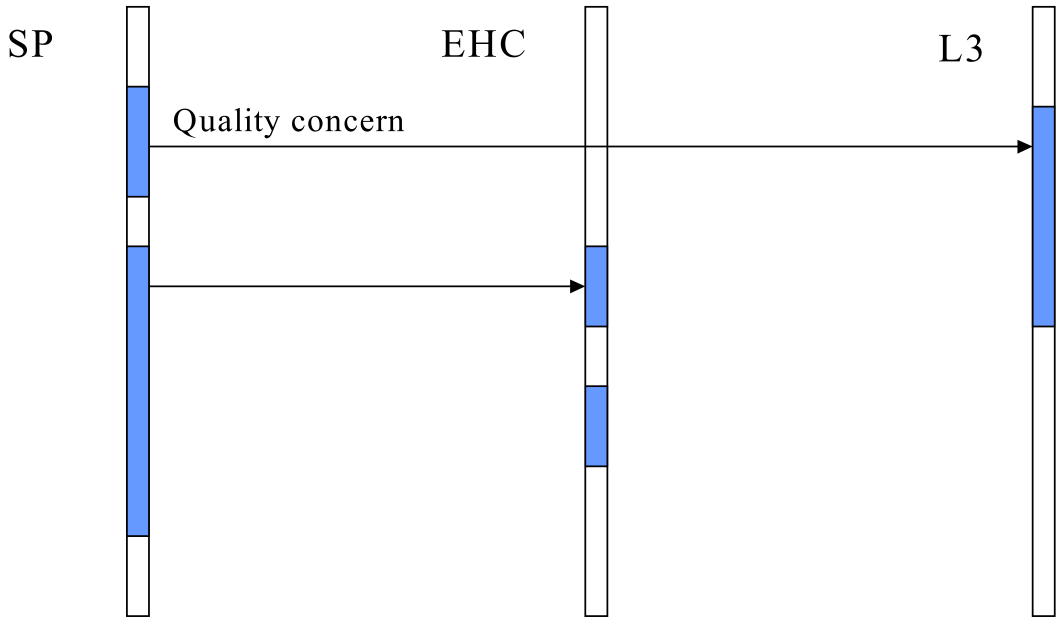


Figure 6.5 Close mode of cross-organizational exception handling

In the close handling mode, a CPR handler template is as follows.

- If (mode is close) signal the exception through exception handling coordinator;
- If (compensation is necessary) execute the compensation scheme;

COORDINATION ANALYSIS

Now we turn our attention to discuss the applicability of exception handlers in these coordination modes, interfaces that should be exposed and correctness issues for possible handlers.

IMMEDIATE MODE

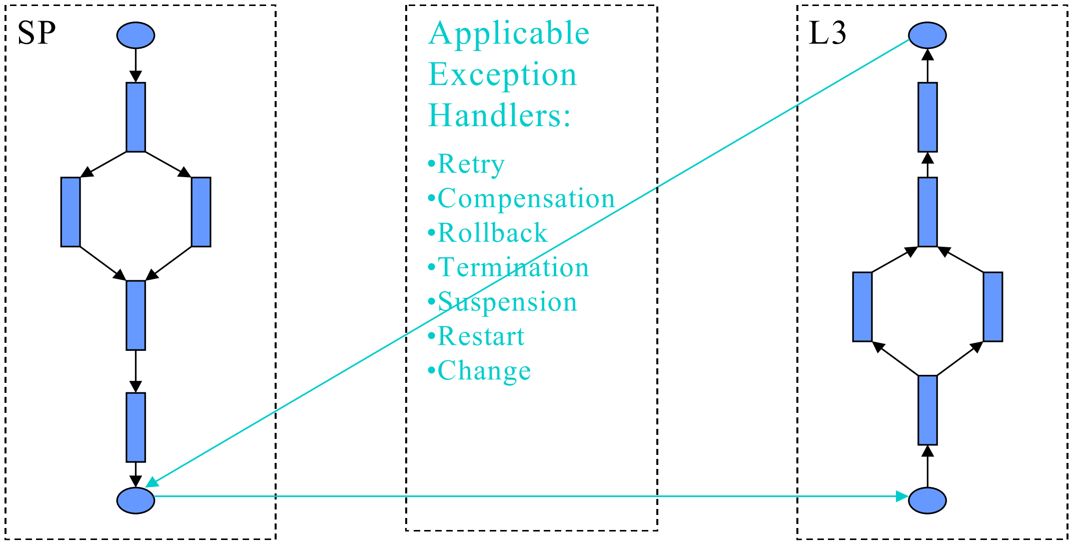


Figure 6.6 Applicable exception handlers in immediate exception handling mode

Cooperating processes must expose their service, monitoring and/or control interfaces to allow immediate exception handling (see Figure 6.6). The selection of specific handlers must ensure the correctness execution of the processes.

INTERFACE EXPOSURE

To get an immediate exception thrown when a request is issued from SP to L3, service requestor in SP can wait on the two-way interface after the request to get the response back. However, this scheme is usually not good for performance, because the waiting halts the system. This is one of the reasons why in OMG's interoperability standard, chained sub-process is proposed. So immediate handling mode usually is suitable for handling exceptions happening in the interconnection establishment stage. When the connection cannot be established, exceptions will be thrown immediately.

Once the service request is routed from service requestor (SP) to service provider (L3), situations are more complicated. For example, the service fulfillment may take time from several minutes to several days, it will cause performance degradation if the service requestor is waiting for the fulfillment to complete or for an exception to occur. It is more efficient to request the service provider to expose monitoring interfaces and control interfaces. So service requestor can monitor the execution status and if an exception is raised, handlers can be selected from the exposed control interfaces or from the service requestor's handler package. However, this is no longer immediate exception handling. We will discuss this in other handling modes.

Correctness issue

In immediate exception handling mode, because the exception is immediately obtained, applicable exception handlers in immediate exception handling mode include retry, compensation, termination and workflow modification and evolution. The exceptions in this mode are handled by using try-catch block.

- Retry. A retry exception handler usually poses no threat to correctness violation. The only concern, a retry operation should not generate an instance with a new instance id.
- Compensation. Usually compensation is built into the process model. When a compensation operation is needed, it helps compensate what has been accomplished. In the immediate handling mode, the service requestor is usually free to choose another process to accomplish its goal, while the compensation is usually not expensive because the service has not been started yet. If the service requestor selects a compensation scheme that is not included in the process model, his/she must ensure that the correctness criteria are not violated.

- Termination. Service requestor (SP) should be able to terminate the process in L3 if it decide not to use that failure process. L3 needs to ensure its process meets the termination requirement and expose necessary interface for SP to do so.
- Workflow modification and evolution. Service requestor (SP) should be able to modify the process in L3 to its benefits through re-configuration or other dynamic change methods. L3 needs to ensure its process meets the various correctness requirements and expose necessary interface for SP to do so.

DEFERRED MODE

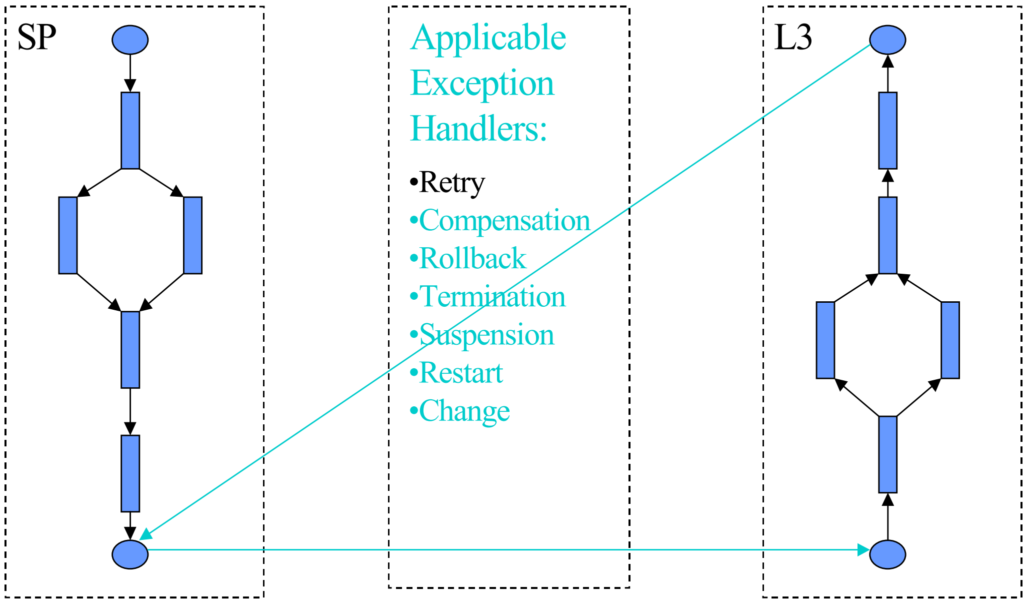


Figure 6.7 Applicable exception handler in deferred exception handling mode

Cooperating processes must expose their service, monitoring and/or control interfaces to allow deferred exception handling (see Figure 6.7). The selection of specific handlers must ensure the correctness execution of the processes.

Interface exposure

In immediate exception handling mode, to get an immediate exception thrown when a request is issued from SP to L3, service requestor SP needs to wait on the two-way interface after the request to get the response back. We have mentioned such a waiting halts the system. In deferred exception handling mode, the service request is already routed from service requestor (Customer) to service provider (SP). The service fulfillment may take time from several minutes to several days. It will cause performance degradation if the service requestor is waiting for the fulfillment to complete or for an exception to occur. Instead, in deferred handling mode, the request requestor (Customer) will expose an interface to let SP to report exceptional situations. So SP can push the exceptional information to service requestor (Customer). Service provider SP needs to expose interfaces to let service requestor to continue the execution.

Correctness issue

In deferred exception handling mode, applicable exception handlers in deferred exception handling mode includes compensation, rollback, termination, execution continuance and workflow modification and evolution. Retry is not appropriate because the requested service has already been fulfilled.

- Compensation. Usually compensation is built into the process model. When a compensation operation is needed, it helps compensate what has been accomplished. In the immediate handling mode, the service requestor is usually free to choose another process to accomplish its goal, while the compensation is usually not expensive because the service has not been started yet. This is not the case in deferred handling mode since the service has already been performed. If the service requestor selects a compensation scheme that is not included in the process model, his/she must ensure that the correctness criteria are not violated.

- Rollback. Service provider SP should allow its service requestor to rollback the service accomplished. It is an expensive choice though to both requestor and provider.
- Termination. Service requestor (Customer) should be able to terminate the process in SP if it decides not to use that process even the request has been fulfilled. If this scheme is used, it usually means service request has low trust on service provider. SP needs to ensure its process meets the termination requirement and expose necessary interface for Customer to do so.
- Workflow modification and evolution. Service requestor (Customer) should be able to modify the process in SP to its benefits through re-configuration or other dynamic change methods. SP needs to ensure its process meets the various correctness requirements and expose necessary interface for Customer to do so.
- Execution continuation. If service requestor (Customer) chooses to continues the execution, it means service provider has gained trust from service requestor. This is the goal why this exception-handling mode is proposed to achieve. Service requestor (Customer) receives an exception, and uses the interface exposed by service provider (SP) to continue the process execution. The key correctness issue here is the live-ness criterion must be ensured.

De-coupled Mode

Cooperating processes must expose their service, monitoring and/or control interfaces to allow de-coupled exception handling (see Figure 6.8). The selection of specific handlers must ensure the correctness execution of the processes.

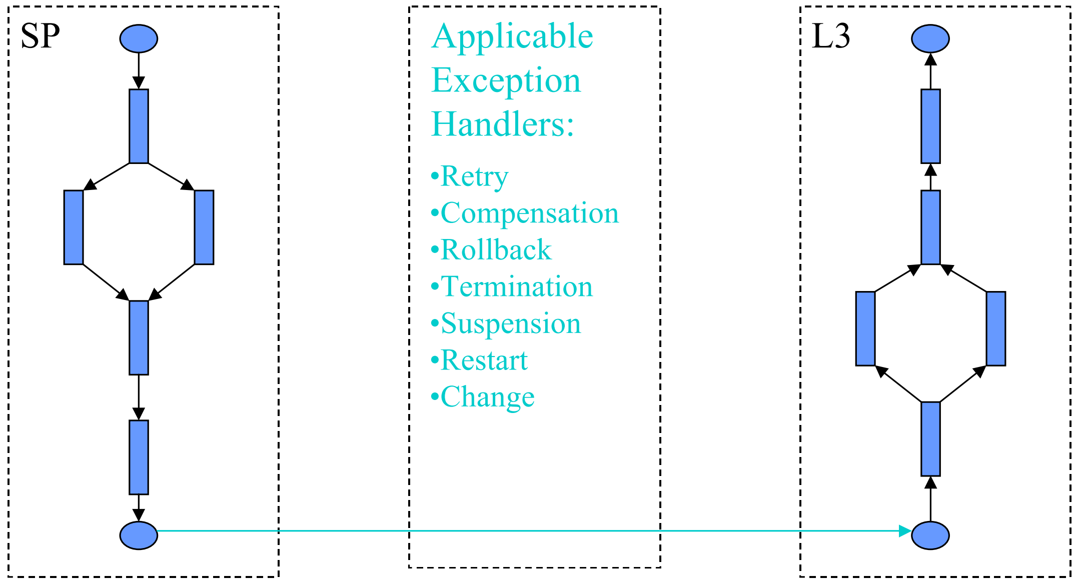


Figure 6.8 Applicable exception handler in de-coupled exception handling mode

Interface Exposure

In immediate exception handling modes, to get an immediate exception thrown when a request is issued from SP to L3, service requestor SP needs to wait on the two-way interface after the request to get the response back (see Figure 6.6). In deferred handling mode, the request requestor (Customer) will expose an interface to let SP to report exceptional situations. So SP can push the exceptional information to service requestor (Customer) (see Figure 6.7). In de-coupled exception handling, there are no interactions after the service request has been routed from service requestor (SP) to service provider (L3). Two processes in these two domains can proceed irrespective of each other. When the requested service being executed in L3 enters exceptional state, i.e. there is an exception thrown for this exception which can not handled locally, the service requestor (SP) should be aware of this exceptional situation. Since both instances expose no interfaces for them to interact, the best chance to resolve the problems relies on the exception handling coordinator (EHC). EHC should expose interfaces for cooperating processes to report exceptions, while it is also aware of the interfaces exposed by the processes. In this way, EHC helps resolve the problems.

CORRECTNESS ISSUE

Applicable exception handlers in de-coupled exception handling mode include retry, compensation, rollback, termination, suspension, restart of tasks, and workflow modification and evolution (see Figure 6.8).

- Retry. EHC (see Figure 6.8) can retry the failed task in the process residing in L3. EHC needs to ensure it generates no instances with new instance ids.
- Compensation. Like in immediate and deferred handling modes, usually compensation is built into the process model. When a compensation operation is needed, it helps compensate what has been accomplished. In the deferred handling mode, EHC should select a compensation scheme that is not included in the process model, it must ensure that the correctness criteria are not violated.
- Rollback. Service provider L3 should allow EHC to rollback what has been accomplished. It is an expensive choice though to both requestor and provider. In this de-coupled handling mode, the process in SP has already proceeded far more than the state to which the process in L3 will be rolled back. EHC must be very careful to ensure the process in L3 can proceed

correctly after the rollback is called. Otherwise the process in L3 might end in deadlock.

- Termination. Like in deferred handling mode, service requestor (SP) should be able to terminate the process in L3 if it decides not to use that process even part of the request has been fulfilled. If this scheme is used, it usually means service request has low trust on service provider or something really bad happens. L3 needs to ensure its process meets the termination requirement and expose necessary interface to EHC so SP can ask EHC to issue the termination command.
- Workflow modification and evolution. Service requestor (SP) should be able to modify the process in L3 to its benefits through re-configuration or other dynamic change methods via EHC. L3 needs to ensure its process meets the various correctness requirements and expose necessary interface EHC.

FREE MODE

Cooperating processes must expose their service, monitoring and/or control interfaces to allow free exception handling (see Figure 6.9). The selection of specific handlers must ensure the correctness execution of the processes.

INTERFACE EXPOSURE

In immediate exception handling modes, to get an immediate exception thrown when a request is issued from SP to L3, service requestor SP needs to wait on the two-way interface after the request to get the response back (see Figure 6.6). In deferred handling mode, the request requestor (Customer) will expose an interface to let SP to report exceptional situations. So SP can push the exceptional information to service requestor (Customer) (see Figure 6.7). In de-coupled exception handling, there are no interactions after the service request has been routed from service requestor (SP) to service provider (L3) (see Figure 6.8). In free exception handling mode, there are possible interactions after the service request has been routed from service requestor (SP) to service provider (L3) (see Figure 6.9). Two processes are able to locate interaction point so through which the information and control can be exchanged. So both instances will expose interfaces to each other or to EHC in order for them to interact. Since this type of interactions is very flexible, both L3 and SP should ensure their processes are able to proceed without the problem of deadlock and abnormal termination.

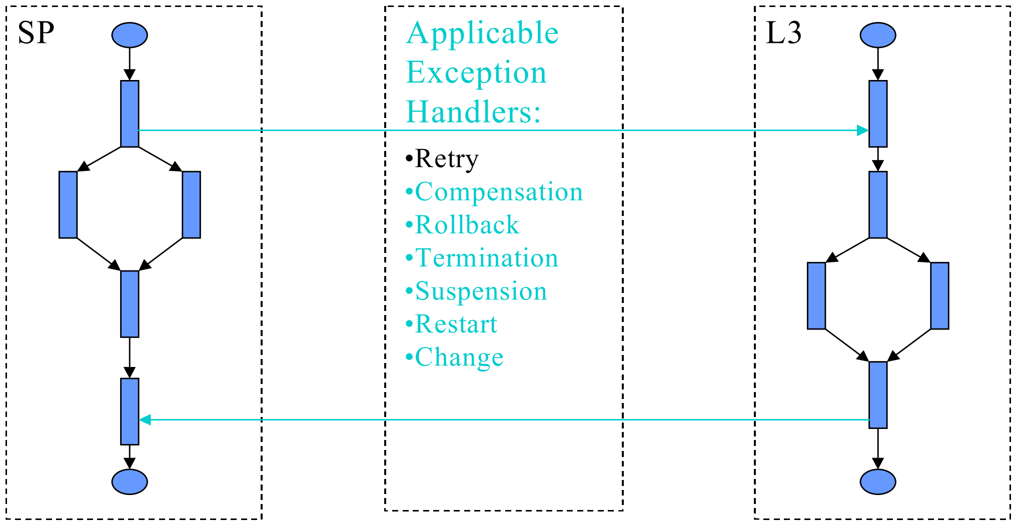


Figure 6.9 Applicable exception handlers in free exception handling mode

Correctness issue

Applicable exception handlers include retry, compensation, rollback, termination, and workflow modification and evolution.

- Retry. Service requestor (SP) is able to retry the failed task in the process residing in L3. L3 and SP need to ensure the retry generates no instances with new instance ids.

- Compensation. In free handling modes, when a compensation operation is needed, it is usually automatically conducted as the scheme is already built into the process mode. Or either process can select compensation scheme it after the exception information is routed to it. Both of the two processes must ensure that the correctness criteria are not violated after compensation scheme is executed.
- Rollback. Processes in both SP and L3 are allowed to rollback what has been accomplished. Like in other handling modes, it is an expensive choice though to both requestor and provider. Like in the de-coupled handling mode, the process in SP has already proceeded far more than the state to which the process in L3 will be rolled back or vice versa. It must be ensured that the process can proceed correctly after the rollback is called.
- Termination. In free exception handling modes, processes in either L3 or SP should be able to terminate the process in other organizations if it decides not to use that process even part of the request has been fulfilled. Both L3 and SP need to ensure their processes meet the termination requirement and expose necessary interface to each other or to EHC so the termination command can be issued.
- Workflow modification and evolution. Processes in both L3 and SP should be able to modify the process in the organization to its benefits through re-configuration or other dynamic change methods. Both L3 and SP need to ensure their processes meet the various correctness requirements and expose necessary interfaces to each other.

CLOSE MODE

Cooperating processes must expose their service, monitoring and/or control interfaces to allow close exception handling (see Figure 6.10). The selection of specific handlers must ensure the correctness execution of the processes.

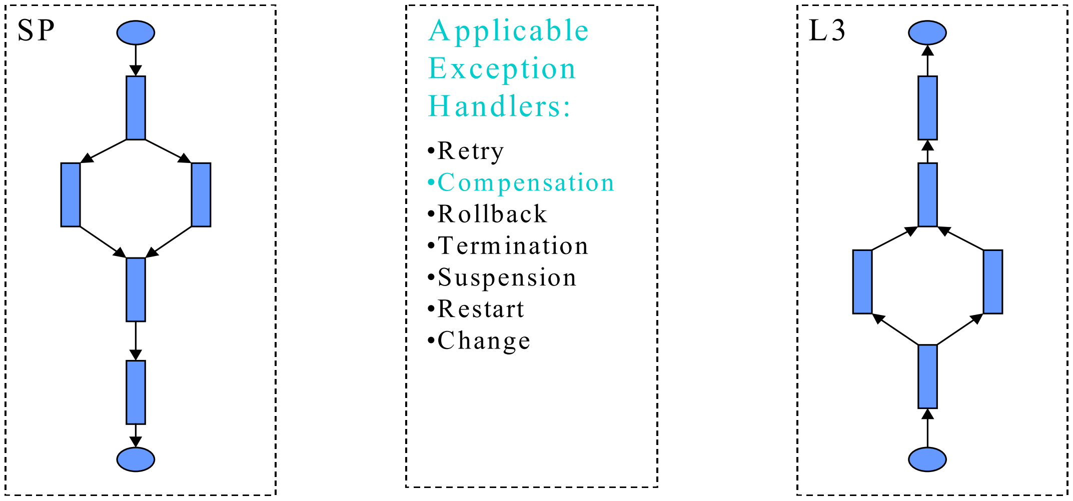


Figure 6.10 Applicable exception handlers in close exception handling mode

INTERFACE EXPOSURE

In close exception handling mode, there are no interactions at all after the service request has been routed from service requestor (SP) to service provider (L3). Two processes in these two domains are independent of each other. When the requested service being executed in L3 enters exceptional state, i.e. there is an exception thrown for this exception which can not handled locally, the service requestor (SP) can't beware of this exceptional situation. Since both instances expose no interfaces for them to interact, the best chance to relieve the problems relies on the exception handling coordinator (EHC). EHC should issue a compensation operation to the process execution environment and record the exceptional situation for later use.

CORRECTNESS ISSUE

In close exception handling mode, applicable exception handlers in close exception handling mode include only compensation. Other handlers are not appropriate because no interactions exist between the two processes. Moreover, in close mode, only environment compensation is possible. Besides compensation, it is possible for the exception handling coordinator to record the case for later use. In free exception handling mode, compensation is used to influence the process execution environment instead to help compensate what has been accomplished.

CHAPTER 7
INTELLIGENT PROBLEM SOLVING

A knowledge-based approach of managing the exception handling knowledge is used in our exception handling system. A case-based reasoning (CBR) mechanism is used to improve the exception handling capabilities. In this approach, information about previous problem solving cases is retrieved to help solve new problems. During the workflow execution, if an exception is propagated to the 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 given by a person will 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 may be made. The exception resolution process is actually the population process of CPR templates. The actual exception resolution performs the following tasks:

- 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 failure points. Human involvement is allowed in determining the rework schemes.

Before we discuss the actual procedures in obtaining the mode, contacting party, interaction point, and the compensation and rework schemes in case adaptation, we are going to turn our attention to the reasoning mechanism we will use in the problem solving process.

REASONING MECHANISM

Two main reasoning mechanisms are used in the intelligent problem solving process. One is case based reasoning (CBR). The other is default reasoning. CBR is the reasoning mechanism that can derive solution by learning from prior experience. CBR is very powerful especially when reasoning is conducted in a single domain on structured knowledge and context information. However, in real applications, like most business applications, these are not the cases. Unstructured data and/or heterogeneity of knowledge and context information are unavoidable. To enhance the problem solving capabilities of CBR, we combine default reasoning with CBR to provide a somewhat more general intelligent problem solving mechanism.

- CBR is the general reasoning mechanism used in the problem solving process. The CBR reasoning steps are generally observed.
- When CBR is applied over unstructured data, or data with missing values, default reasoning is applied to pre-process the unstructured data. The structured output will be supplied to CBR reasoning engine.

CASE BASED REASONING

The CBR based approach models how reuse of stored experiences contributes to expertise [Aamodt 1994]. In this approach, new problems are solved by retrieving stored information about previous problem solving steps and adapting it to suggest solutions to the new problems. The results are then added to the case repository for future use. A case conceptually consists of three parts: problem, solution, and effect as illustrated in Figure 7.1. Problems will be obtained through user or system input. There are solution candidates for these problems. The selection of the output solution will be based on the analysis of the effects of those solutions.

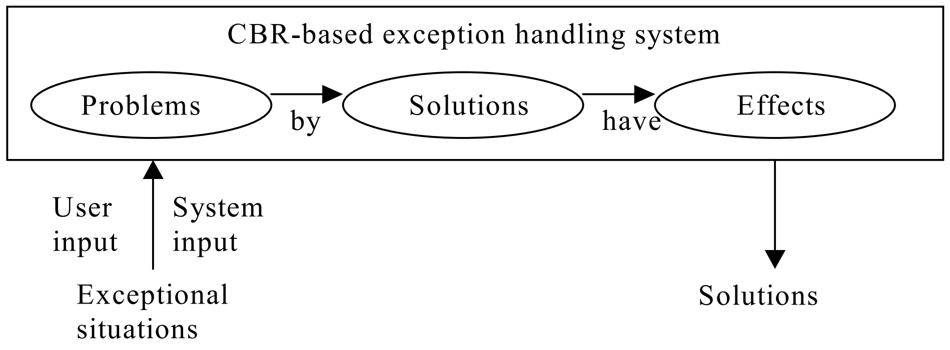


Figure 7.1 Exception handling via CBR based mechanism

```
Input: exception
Job:
Retrieving cases similar to the problems encountered;
Reviewing retrieved cases;
Adapting retrieved cases to the new situation;
Writing back the new generated cases;
Output: solution
```

Figure 7.2 Case acquiring algorithm in pseudo code

As shown in Figure 7.3, the case-based reasoning architecture consists of the following components.

- Abstractor abstracts the exceptional situation based on ontology.
- Retrieval component retrieves related cases from case repository.
- Analyzer analyzes the solution in the cases, and tries to derive a possible solution.
- Retainer writes back the new case into the repository and outputs the solutions.
- Case repository is the place where cases are stored.
- Editor provides GUI interface to modify cases.
- Browser provides GUI interface to allow users to browse the case repository.
- Explainer explains the cases to users through GUI interface. This component facilitates users to understand cases, such as what the cases are, why the solutions are effective, and their effects.
- Ontology & Concept component manage the concepts in multiple domains.
- Similarity Measure component provides similarity measure algorithms during case retrieval. Cases are often heterogeneous. Different cases need different similarity measure algorithms.

When exceptions occur, the abstractor identifies the exceptional situation via input (system input or user input). It extracts information from input. Output of the abstractor is fed to retrieval component. The retrieval component retrieves similar cases for analysis by the case analyzer. An acceptable solution may be derived at this stage. That is, the solution of a similar case can be applied. The new derived case will be forwarded to the retainer. Retainer may write back the new case and will construct solutions and supply it to systems or users. In some situations no similar cases can be found or the analyzer is not confident about the similarity measurement. That is, the value of the similarity measure is too low. In such situations, the case editor will allow users to derive acceptable solutions. During the interaction, the explainer helps users to understand the cases.

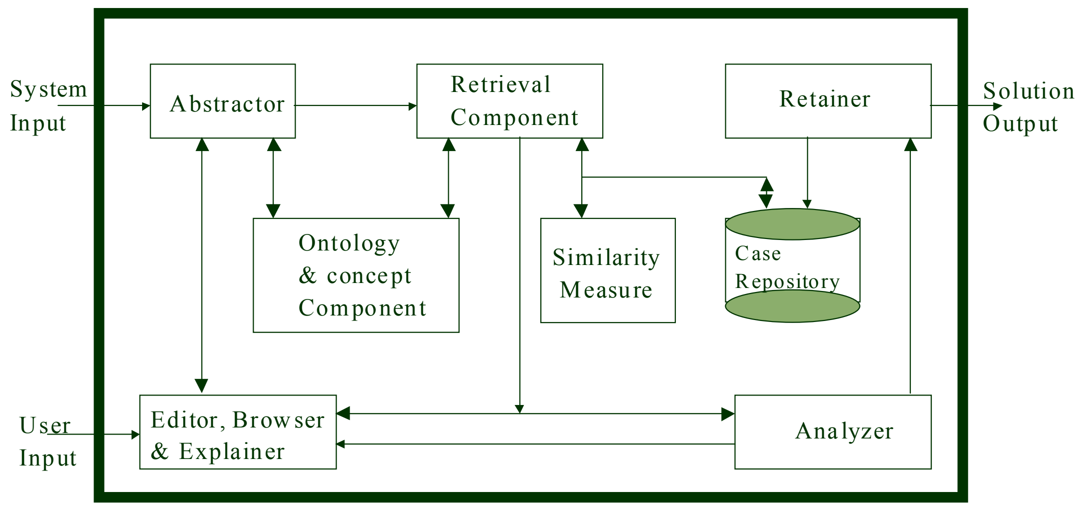


Figure 7.3 CBR concept architecture

CONCEPT BASED CASE MANAGEMENT

To realize the CBR based exception handling, we propose a specific technique: concept based case management. Since this case-based reasoning system will be applied in various domains (such as healthcare, telecommunication, finance, etc.), we use ontology to describe the concepts used in various domains in the form of concept trees. In this CBR based exception-handling approach, we usually need to acquire, represent, index and adapt existing cases to effectively apply the case-based reasoning process. In our approach, we acquire new cases by learning through exceptions. That is, we create new cases when exceptions occur. We use this concept-based case management for the following activities:

- Case acquiring. There are two ways to create new cases. First, cases are created based on knowledge obtained in expert survey, interview, etc. Secondly, cases are created in the exception resolution processes.

- Case retrieval. We use concepts to organize cases. The retrieval of similar cases is based on concept-based similarity measurement.
- Case adaptation. Adaptation increases the usability of existing cases. A scheme of pattern guided case adaptation is proposed.

CASE ACQUIRING

Knowledge obtained through expert surveys and interview is represented as cases. These cases are stored in the case repository. During the exception resolution, these cases are consulted in order to find exception solution. New cases can be created during the resolution process.

A conceptual CBR based problem-solving architecture, shown in Figure 7.4, is used to realize this problem-solving methodology. The input to the CBR based problem-solving algorithm will be supplied either by a human or by a workflow system component. This input will trigger the CBR problem solver to retrieve cases stored in case repository in handling similar exception situations. Retrieved cases will be analyzed according to the new situation. To reuse the retrieved cases, adaptation must take place. The adapted cases are written back into case repository.

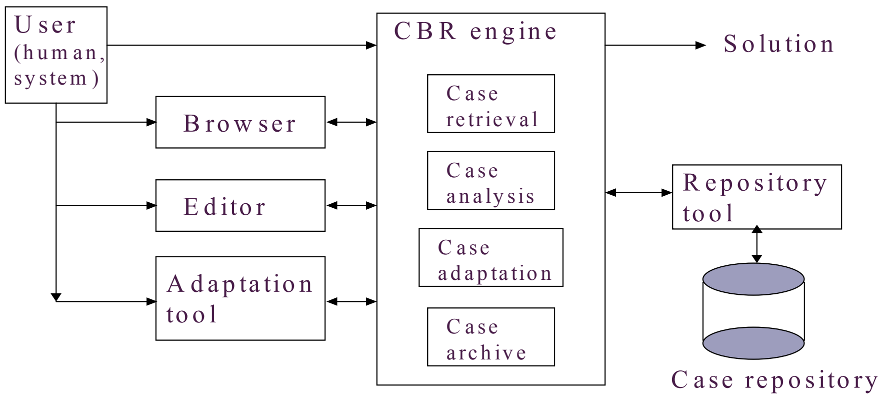


Figure 7.4 Conceptual architecture of CBR based problem solver

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 [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 [Luo et al 2000]. 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 7.5. 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 7.5:

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

Figure 7.5 Similarity equation for two cases

As shown in Figure 7.5,

- w_i is the weight for ith attribute.
- a_i is ith attribute in case a.
- b_i is the ith 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 [Luo et al 2000] 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.

CASE ADAPTATION

There are two main approaches to realizing case adaptation:

- Problem adaptation: One way to adapt a case is to enhance the partial description about the problem. Another way is to substitute conception realization in a case.
- Solution adaptation: There are two ways associated with solution adaptation [Aamodt 1994]: (1) reuse the past case solution (transformational reuse), and (2) reuse the past methods that constructed the solution (derivational reuse).

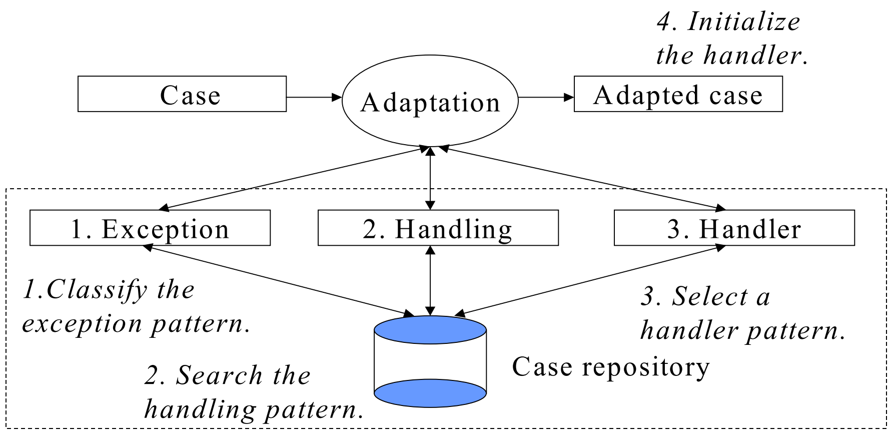

4. Initialize the handler.
Case
Adaptation
Adapted case
1. Exception
2. Handling
3. Handler
1.Classify the exception pattern.
3. Select a handler pattern.
Case repository
2. Search the handling pattern.

Figure 7.6 Pattern guided case adaptation

There might be combinations of the above two approaches. However, a case can be used without any modifications, which is usually called NULL adaptation. Partial matching during case retrieval is also one way to realize case adaptation. There are usually two approaches to case modifications: parameterized modification and substitution modification. Modification of a case by substitution usually results from the fact that information in the action information block of the case is not applicable or available.

We propose a pattern guided case adaptation scheme (see Figure 7.4). There are four steps in the adaptation process in this pattern guided adaptation scheme. The process is really the population process of the CPR handling template.

- Classifying the exception pattern. At this step, the exception pattern will be identified. If it is a new pattern, it will be added to exception pattern repository.
- Searching the handling pattern. Once the exception pattern is determined, a search will be conducted for the handling pattern. At this step, the exception handling coordination mode will be determined. Contacting party as well as interaction point is also determined by analyzing the interactions among business processes.
- Selecting a handler pattern. A handler pattern will be selected based on the search result from step 2. The compensation scheme as well as the rework scheme will be determined.
- Initializing the handler. The CPR handling template will be populated. An adapted case is created.

CHAPTER 8

METEOR ORBWORK WORKFLOW MANAGEMENT SYSTEM

Our exception handling system is being implemented for the METEOR workflow management system. The METEOR project is represented by both the research system [METEOR], and a suite of commercial offering [Infocosm] that provides an open-systems based high-end workflow management solution as well as an enterprise application integration infrastructure.

METEOR ARCHITECTURE

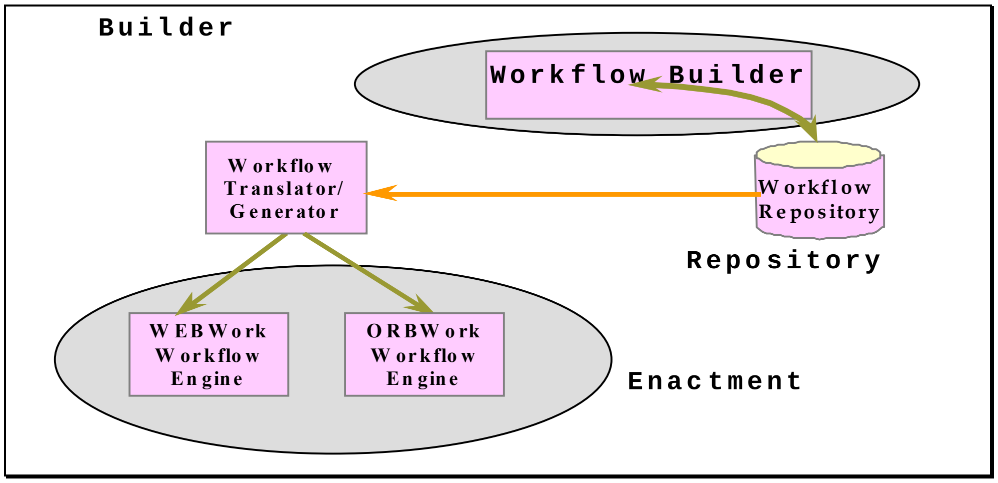


Figure 8.1: METEOR Architecture

METEOR's architecture (see Figure 8.1) includes a collection of four services: Builder, Repository, Enactment, and Manager. Enactment includes two services- ORBWork [Kochut et al. 1999] and WebWork [Miller at al. 1998]. Both ORBWork and WebWork use fully distributed implementations. WebWork, an entirely Web-based enactment service, is a comparatively light-weight implementation that is well-suited for some types of enterprise workflow process applications that involve limited data exchange and do not need to be dynamically changed. ORBWork is targeted for more demanding, mission-critical enterprise applications requiring high scalability, robustness and dynamic modifications.

WORKFLOW BUILDER SERVICE

This service consists of a number of components that are used to graphically design and specify a workflow, in some cases leaving no extra work after a designed workflow is converted to a workflow application by the lightweight code generator [Kang et al 1999]. Its three main components are used to specify the entire map of the workflow, data objects manipulated by the workflow, the details of task invocation, as well as security domain respectively [Kang et al 1999]. This service supports modeling of complex workflows consisting of varied human and automated tasks in a conceptual manner using easy to use tools. In particular, the designer of the workflow is shielded from the underlying details of the infrastructure or the runtime environment. At the same time, very few restrictions regarding the specification of the workflow are placed on the designer.

The workflow specification created using this service includes all the predecessor-successor dependencies between the tasks as well as the data objects that are passed among the different tasks. It also includes definitions of the data objects, and the details of the task invocation details. This service assumes no particular implementation of the workflow enactment service (runtime system). Its independence from the runtime supports separating the workflow definition from the enactment service on which it will ultimately be installed and used. Workflow process definitions are stored in the workflow repository.

Initial work on workflow designer can be found in [Lin 1997, Zheng 1997].

WORKFLOW REPOSITORY SERVICE

The METEOR Repository Service is responsible for maintaining information about workflow definitions and associated workflow applications. The graphical tools in the workflow builder service communicate with the repository service and retrieve, update, and store workflow definitions [LSDIS 2000]. The tools are capable of browsing the contents of the repository and incorporating fragments (either sub-workflows or individual tasks) of already existing workflow definitions into the one being currently created. The repository service is also available to the enactment service (see below) and provides the necessary information about a workflow application to be started.

Initial work in repository service can be found in [Yong 1998].

WORKFLOW ENACTMENT AND MANAGEMENT SERVICES

The task of the enactment service is to provide execution environment for processing workflow instances. At present, METEOR provides two different enactment services: ORBWork and WebWork. Each of the two enactment services has a suitable code generator that can be used to build workflow applications from the workflow specifications generated by the building service or those stored in the repository. In the case of ORBWork, the code generator outputs specifications for task schedulers (see below), including task routing information, task invocation details, data object access information, user interface templates, and other necessary data. The code generator also outputs the code necessary to maintain and manipulate data objects, created by the data designer. The task invocation details are used to create the corresponding “wrapper” code for incorporating legacy applications with relative ease. Details of code generation for WebWork are presented in [Miller et al. 98]. The management service support monitoring and administering workflow instances as well as configuration and installation of the enactment services.

Initial work in enactment service can be in [Das et al. +97].

ORBWORK ENACTMENT SYSTEM

The current version of ORBWork, the one of the two implementations of the METEOR enactment services has been designed to address a variety of shortcomings found in today's workflow systems by supporting the following capabilities:

- provide an enactment system capable of supporting dynamic workflows,
- allow significant scalability of the enactment service,
- support execution over distributed and heterogeneous computing environments within and across enterprises,
- provide capability of utilizing or integrating with new and legacy enterprise applications and databases in the context of processes,
- utilize open standards, such as CORBA due to its emergence as an infrastructure of choice for developing distributed object-based, interoperable software,
- utilize Java for portability and Java with HTTP network accessibility,
- support existing workflow interoperability standards, such as JFLOW [JFLOW] and SWAP [SWAP] (initial work can be found in [Ketan 1999]), and
- provide standard Web browser based user interfaces, both for the workflow end-users/participants as well as administrators of the enactment service and workflows.

Scalability of the enactment system is becoming increasingly important for enterprises that wish to entrust their workflow management system with mission-critical processes. The number of concurrent workflows, the number of instances of the workflows processed during a given time period, and the average number of tasks in a workflow, all have an impact on the architectural issues.

We have leveraged the functionality offered by Iona's OrbixWeb and Name Service that allow us to place various components of the enactment service or other run-time components of the workflow instances, such as task schedulers, task managers, data objects, and even actual tasks on different hosts, at the same time providing transparency of their locations.

The ORBWork scheduler and its supporting components have been designed in such a way that the enactment service can be used to support a variety of dynamic changes both to the workflow schema and to the individual workflow instances. The fully distributed scheduler (described later) maintains the full workflow specification. The workflow administrator can easily modify the workflow schema at runtime by acquiring new information from the workflow repository, or even by modifying the specification by direct interaction with the scheduler.

ORBWork provides a fully distributed, scalable enactment system for the METEOR workflow management system. The enactment system has been implemented to support workflows in heterogeneous, autonomous and distributed (HAD) systems. It utilizes the World Wide Web in providing a consistent interface to end-users and workflow administrators from commonly available Web browsers, and also utilizes the HTTP protocol for distribution of task definitions and task routing information.

ORBWORK ARCHITECTURE

ORBWork's architecture includes the scheduler, workflow specification repository, workflow manager, and the monitor. An overview of the ORBWork system organization is depicted in Figure 8.2.

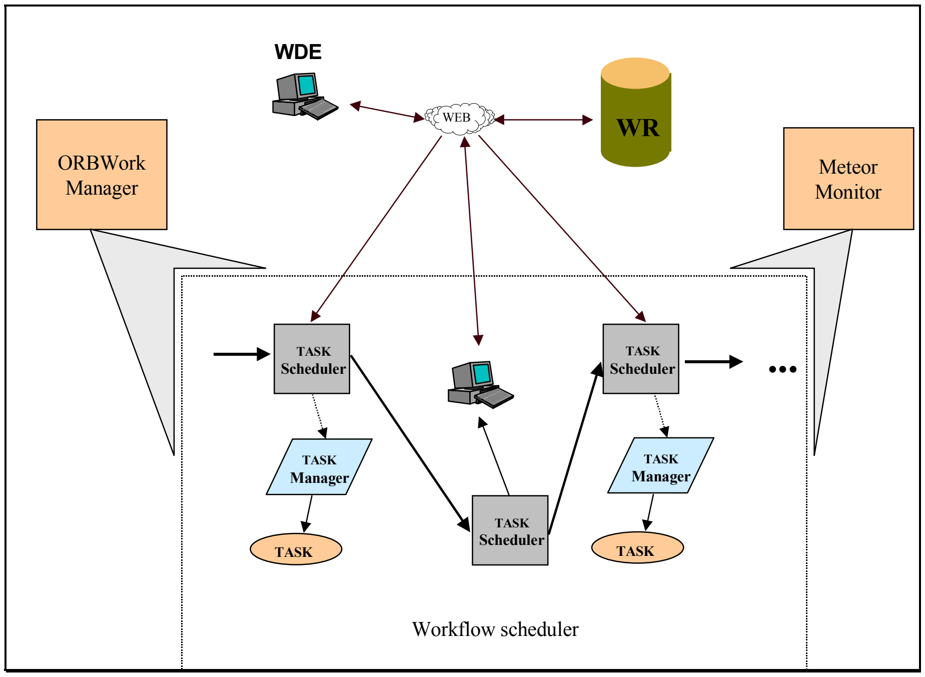


Figure 8.2: ORBWork organization

The *scheduler* accesses workflow specifications through the HTTP protocol, directly from the repository. The *monitor* records all of the events for all of the workflows being processed by the enactment service. It provides a user interface for the workflow administrator, who can access the information about all of the current workflow instances. The *workflow manager* is used to install new workflow processes (schemas), modify the existing processes, and keep track of the activities of the scheduler. The workflow administrator, using the available interface, controls the existing workflows as well as controls the structure of the scheduler. The structure of the scheduler can be altered by adding more resources, or by migrating fragments of the scheduler to other hosts, for example with lower processing loads. Some schedulers may be replicated, in case the load of workflow instances is too high for a host running just a single scheduler.

ORBWork's scheduler is composed of a number of small schedulers, each of which is responsible for controlling the flow of workflow instances through a single task. The individual schedulers are called *task schedulers*. In this way, ORBWork implements a fully distributed scheduler in that all of the scheduling functions are spread among the participating task schedulers that are responsible for scheduling individual tasks. In this sense, the ORBWork scheduler is composed of a network of cooperating task schedulers. Each task scheduler controls the scheduling of the associated task for all of the workflow instances "flowing" through the task. Each task scheduler maintains the necessary task routing information and task invocation details (explained later).

As a workflow instance progresses through its execution, individual task schedulers create appropriate task managers that oversee execution of associated tasks. Each workflow instance receives its own task manager, unless the task has been designed to have a worklist, in which case all of the instances are processed by the same task manager.

A workflow is installed by first creating an appropriate workflow context in the Naming Service. (The context is used for storing the object references for all of the participating components.) Then the installation continues by activating and configuring all of the necessary task schedulers and registering them with the Naming Service. All of the component task managers are also registered with the Interface Repository of the underlying ORB.

ORBWORK SCHEDULER

ORBWork utilizes a fully distributed scheduler in that the scheduling responsibilities are shared among a number of participating *task schedulers*, according to the designed workflow map. Each task scheduler receives the scheduling specifications at startup from the Workflow Repository. Each set of task specifications includes the input dependency (input transitions), output transitions with associated conditions, and data objects sent into and out of the task. In case of the human task (performed directly by end-users), the specifications include an HTML template of the end-user interface page(s). In case of a non-transactional automatic task (typically performed by a computer program), the specifications also include a task description and the details of its invocation. Finally, in case of a transactional task, the specification includes the details of accessing the desired database and the database query.

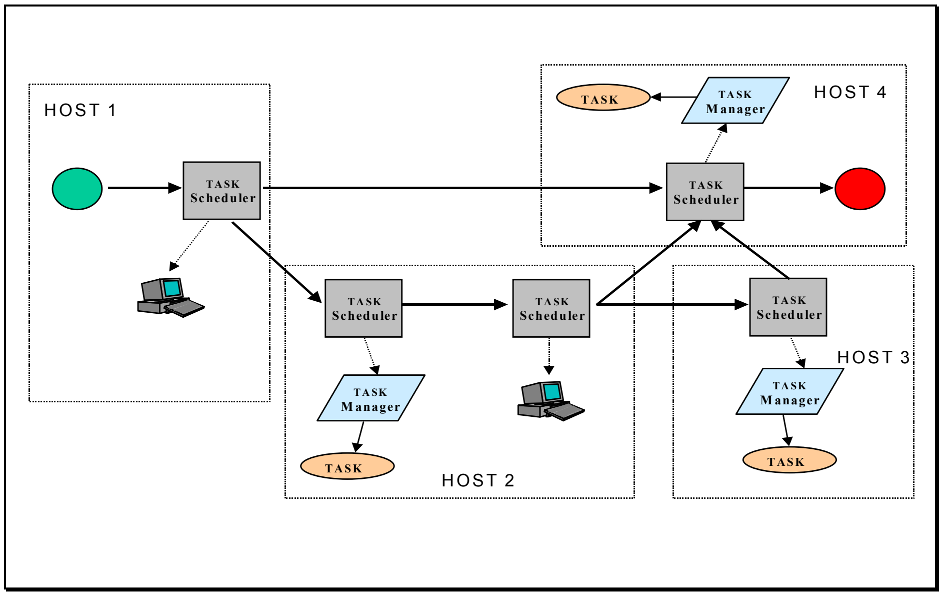


Figure 8.3 ORBWork's Distributed Scheduler

When a task is ready to execute, a task scheduler activates an associated task manager. The task manager oversees the execution of the task itself. Figure 8.3

presents a view of the ORBWork's distributed scheduler. Note that scheduling components and the associated tasks and task managers are distributed among four different hosts. The assignment of these components to hosts can be modified at runtime by the workflow administrator.

The partitioning of various components (scheduler's layout), including task schedulers, task managers and tasks, among the participating hosts is flexible. An ORBWork administrator may move any of the components from one host to another. In the fully distributed layout, it is possible to place all of the workflow components on different hosts.

Each task scheduler provides a well-constrained subset of the HTTP protocol, and thus implements a lightweight, local Web server. This enables an ORBWork administrator to interact directly with a selected task scheduler and modify its scheduling specifications from a common Web browser. It also enables the end-user to access workflow instances residing on the task's worklist. This set up naturally supports a mobile user.

EXCEPTION HANDLING

ORBWork supports exception handling as well. Exception handling in ORBWork is described in METEOR model 3 [METEOR model 3]. During the workflow execution, a number of undesirable events may occur. For example, one of the hosts used by a workflow application may crash, a network connection may go down, or possibly one of the tasks in the workflow application may detect an exceptional condition, specific to the task itself. In METEOR model3, an undesirable event may fall into the category of a workflow management system-specific, or workflow application-specific. An exception that is system specific is called a system exception, while a workflow-specific exception is called an application exception. In METEOR model 3, system exceptions may occur during the execution of every workflow, while application exceptions are restricted to specific applications.

In METEOR model 3, an exception is viewed as an occurrence of some abnormal event that the underlying workflow management system can detect and react to. Any abnormal event that is not detected by the enactment system will not be considered to be an exception in METEOR model 3. This position is obtained from the implementation or enactment point of view. From the modeling point of view, this abnormal event is still an exception. Instead, in METEOR model 3, this abnormal event will be considered as external exception signal. 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 enactment (and/or the workflow application may be notified externally of such an event by receiving an exception signal. An exception signal may be sent by an external program (for example, a database trigger), or manually, by a workflow administrator.

An exception handler is a description of action(s) that the workflow enactment system, or possibly a workflow application, is going to perform in order to respond the exception.

In order to provide an exception handler, we must consider:

- Location of control at the time of exception, and
- Exception type.

Placement of control forms a hierarchy of processing components according to the control structure (see Figure 8.4) in METEOR model 3. This hierarchy may be extended even further, if a workflow application has a hierarchy of tasks with network realizations (the actual task implementations). An exception may occur at any time, while control is within one of the components in the hierarchy.

Since an exception is most likely to occur while a workflow application is running one of the task realizations, the most typical form of an exception handler is a failure transition. A failure transition is a transition in a network that leads from a failure state of some task. A failure transition is associated with a specific exception and is designed to handle the exception, or any of its descendant exceptions.

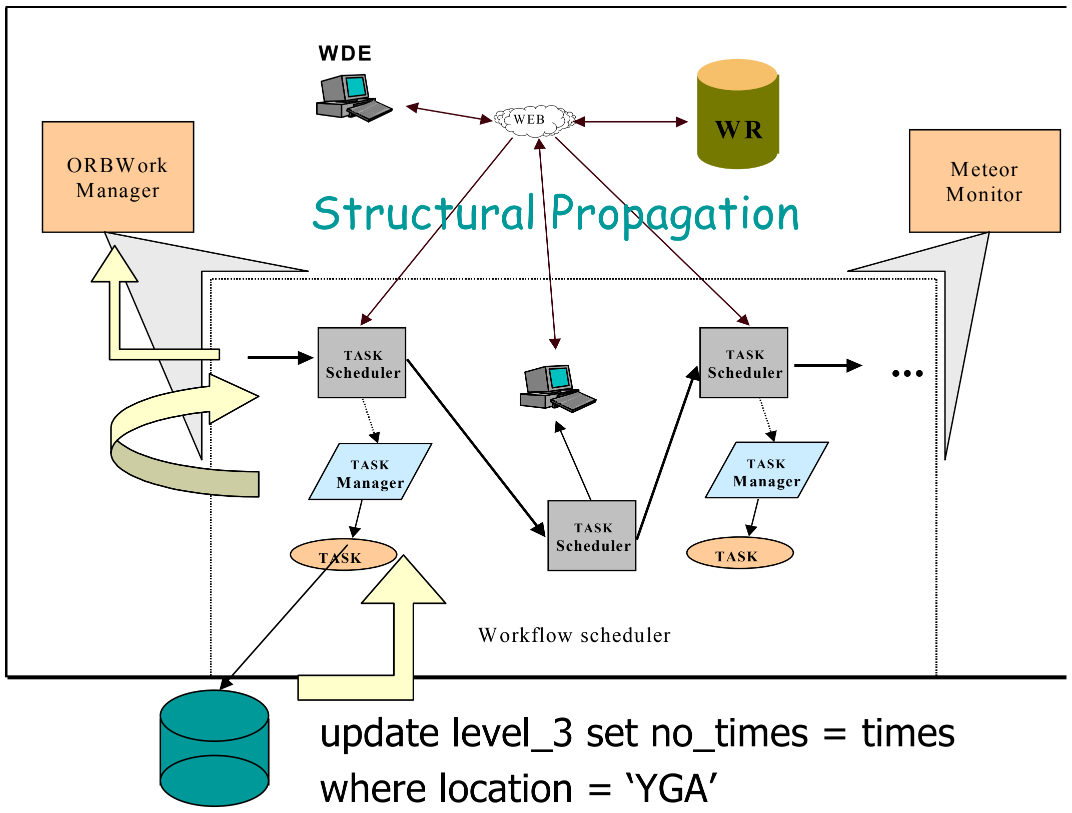


Figure 8.4 Exception propagation in METEOR ORBWork system

If an exception occurs while a task manager is attempting to execute a task realization, the task manager itself may have handler for this exception. The handler in this case may involve a number of retries. Eventually, if all retry attempts are unsuccessful, the task manager may decide to re-throw the exception.

A competence-driven exception handling is adopted (see Figure 8.4). 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 resides. If the component is competent to handle the exception, that is it has at least one handler for the given exception type, it handles the exception. On the other hand, if the component does not have the competence for the exception, the exception is passed to the next higher level in the competence hierarchy.

If a component has been designed to handle a given exception, it does so using its own handlers. A component may also decide to send the exception to the higher level in the component hierarchy. If a component has not been designed for handling of the exception, the exception is passed onto the higher level by default.

To be more specific, if an exception occurs during an invocation of a task realization, (e.g., database update task in Figure 8.4), its parent task enters its fail state and a suitable exception object is made available for scheduling purposes. This is analogous to the exception being thrown by the task.

If the task is part of a network, which is a realization of a higher-level task, the network may be competent to handle the thrown exception (note that even at task manger may be competent to handle an exception). That is, there may be a failure transition set for the given exception. Such a transition is called a handler for the exception. However, if the parent network is not competent to handle the thrown exception, the whole parent network fails and re-throws the same exception to its own parent, which may have the competence necessary for handling the given exception.

Since the exceptions form a hierarchy, a given task may have more than one suitable handler (for example, for a specific exception and for one of its predecessors in the hierarchy). In this case, the most specific handler is used.

If the task representing the whole workflow fails, the whole workflow fails and the exception is passed on to the workflow scheduler. If the thrown exception is one of the exceptions for which the scheduler is competent, it is then handled accordingly to its type by the scheduler. Otherwise, the scheduler re-throws the exception to the workflow manager (see Figure 8.4).

The workflow runtime system (workflow manager) has default handlers for all the workflow exceptions. In other words, the workflow manager is competent to handle any exception, whether system or application-defined. The handlers invoke the recovery mechanism, whenever necessary and possible.

If an exception is detected while a task manager is attempting to run a task realization (for example, a computer program), the task manager itself may be competent to handle the exception with one of its internal handlers. Such handlers typically involve retrying the task realization a number of times, before re-throwing the exception to the higher level component.

It is possible that an abnormal event cannot be detected by any of the components of the workflow system. For example, a currently running workflow instance is in violation of a newly introduced business policy. It might be the case that the workflow runtime should force a workflow instance to fail the next task and continue its execution following one of its alternate paths. However, at this time neither the workflow application nor the workflow runtime system is aware of the abnormal event. Such an abnormal event is called an external fault.

An exception signal may be sent to the workflow system in order to make an external fault known to the workflow. The signal may be sent by an external entity, such as a workflow administrator, or a separate program. The workflow administrator may decide to force a particular workflow instance to fail one of its currently running tasks, and send a suitable exception signal to the task manager. Similarly, a process monitoring database updates may trigger sending an exception signal to the workflow manager. Another example of an exception signal may be a TimeElapsed signal sent to a task manager by the workflow runtime timer. The task manager may then issue a TimeElapseException as the result of a task not completed within a specified time.

The workflow system received the exception signal and continues its operation as if the exception was detected by the workflow itself.

ORBWORK IMPLEMENTATION

One of the most important considerations while designing the ORBWork workflow management system has been its flexible and easily modifiable distributed architecture. The current version of the system has been implemented in Java and OrbixWeb 3.2, Iona's CORBA system with Java binding. In addition, Iona's Naming Service has been utilized as a way of providing location transparency for all of the ORBWork components.

Using CORBA, and especially Iona's OrbixWeb and Naming Service, as the underlying middleware system offers a number of advantages for implementing a distributed workflow enactment system. In addition to the obvious features provided by CORBA, ORBWork relies on a number of specific services that proved extremely useful in implementing ORBWork.

All of the ORBWork components are implemented as CORBA objects. ORBWork relies on the Orbix Activator to start the necessary server when its functions are necessary for the activities of the distributed scheduler and also shutdown the servers once no services have been requested within a specified time interval. In this way, certain portions of a large, distributed workflow (for example those less frequently used) may become inactive, reducing the overhead on the host systems to the necessary minimum.

Task schedulers

A task scheduler is implemented as a CORBA object. The IDL interface presented to clients (other task schedulers and other ORBWork components) enables them to invoke various scheduling functions according to the currently loaded specifications. The interface also enables dynamic modifications of the scheduling specifications by reloading from the specification server (repository) or by a direct modification of the specification within the task scheduler.

A task scheduler relies on Orbix Name Service to locate its successors. This enables the ORBWork administrator to dynamically reconfigure the runtime layout of the scheduler by shifting some components between hosts, without introducing any changes to the remaining task schedulers, or workflow instances administered by them.

ORBWork uses the object loader capability supported by OrbixWeb to save/restore the state of a task scheduler. The state includes the necessary information about forthcoming instances (those with still unfulfilled input dependency) and those already on the worklist. As the CORBA object representing a task scheduler is activated (because one of its task predecessors attempts a transfer of the next workflow instance), the necessary scheduling data is automatically reloaded.

Task managers

Task managers control execution of all non-human tasks (human tasks have no associated task managers). Depending on the task type, a task manager is classified as non-transactional or transactional, and is implemented as a CORBA object. A task manager's IDL interface allows it to be invoked by the corresponding task scheduler. Once activated, the task manager stays active until the task itself completes or generates an exception. Once the task has completed or terminated prematurely with a fault, the task manager notifies its task scheduler. The task scheduler then continues the flow of the workflow instance.

Orbix Activator automatically activates the task manager, only when needed. The communication between the task scheduler and the associated task manager is accomplished by asynchronous (one way) method calls.

A transactional task manager uses JDBC to access the requested data source. Currently, ORBWork provides specific task managers for accessing Oracle and Mini SQL databases, as well as one for the Open JDBC driver from I-Kinetics. The last of the mentioned task managers allows a uniform access to a wide variety of database management systems (including those on mainframes) from a single task manager.

DATA OBJECTS

Data objects are implemented as CORBA objects, providing an IDL interface for accessing all of the defined attributes and methods. As in the case of a task scheduler, the data object implementation uses the object loader to load and save the state of each data object. The CORBA server hosting the data objects is automatically shut down if no data read/write requests arrive within a specified time period, and the dynamic loader saves the state of the object.

As task schedulers implement flow of control within a workflow instance, data objects must be made available at the successor tasks. Instead of the whole object, only the object reference is sent to the task scheduler. When preparing to run the task, the task scheduler accesses the necessary data object(s) (using the Dynamic Invocation Interface) and extracts the relevant attribute values.

ORBWORK SERVERS

Typically, a single ORBWork host runs a number of task schedulers, each of which is implemented as a separate CORBA object. A CORBA object must reside within a CORBA server that typically runs as a single operating system process. In order to save computer resources, a group of ORBWork task schedulers may be placed within a single CORBA server that functions as an ORBWork server. Each ORBWork server is designed to control any number of task schedulers.

A workflow installed on the ORBWork enactment system may utilize any number of heterogeneous hosts (of course, OrbixWeb must be available on each one of them; clients/browsers may be used anywhere). Each of the hosts may have any number of ORBWork servers. However, the most common approach is to keep the number of ORBWork servers close to the number of available processors. Nowadays, some of the available Java virtual machines are able to take advantage of the available processors to run threads. Since the implementation of an ORBWork task scheduler is multithreaded, the question of the number of ORBWork servers may be less critical in that if all of the schedulers are placed within a single server, the schedulers will be able to utilize all of the available processors.

ORBWORK MANAGER

The ORBWork Manager is used to install workflows (schemas) and activate all of the necessary task schedulers. In addition to registering with Orbix Name Service, each task scheduler registers with ORBWork Manager and notifies it of its precise location. In addition, since each task scheduler provides a subset of the HTTP protocol, the scheduler notifies the ORBWork Manager of the precise URL address that the end users and the administrator can use to interact directly with it. The URL address is created when the scheduler is initially installed and it contains the port number that has been assigned to the HTTP server.

The manager is implemented as a CORBA object. It has an IDL interface that allows ORBWork clients to install and administer a workflow (schema) as well as create workflow instances. The manager provides an HTTP protocol, so that the same administrative functions can be performed via the Web, from a common browser.

In order to provide an easy access to task schedulers, the ORBWork Manager also functions as a URL redirector, when an end-user wishes to access her task's worklist. This is necessary since the port number on which the task scheduler's HTTP server is listening is assigned by the system at the time the task scheduler is activated. The port number is not fixed and cannot be known beforehand.

It is important to note that the role of the ORBWork Manager is necessary only at the time a new workflow is installed or modified, or when an end-user is connecting for the first time to his designated task. The manager does not participate in any task scheduling activities.

CHAPTER 9

EXCEPTION HANDLING SYSTEM

Workflow systems need exception detection and handling mechanisms to satisfy reliability requirements [Luo et al 2000]. In fact, an exception is considered a learning opportunity for the workflow systems. It signals the abnormal situations of workflow and workflow execution. During the workflow execution, runtime checking along with those predefined exceptions will be used to detect abnormal situations. However, when such exceptions occur, necessary actions will be taken. If an exception occurs, acceptable exception handler, e.g., recovery activities will be called to handle such exceptional situations. In case an exception can not be handled locally, the case repository will be consulted. Solution contained in the similar case retrieved will be used to handle the exceptions.

SYSTEM ARCHITECTURE

The exception handling system 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.

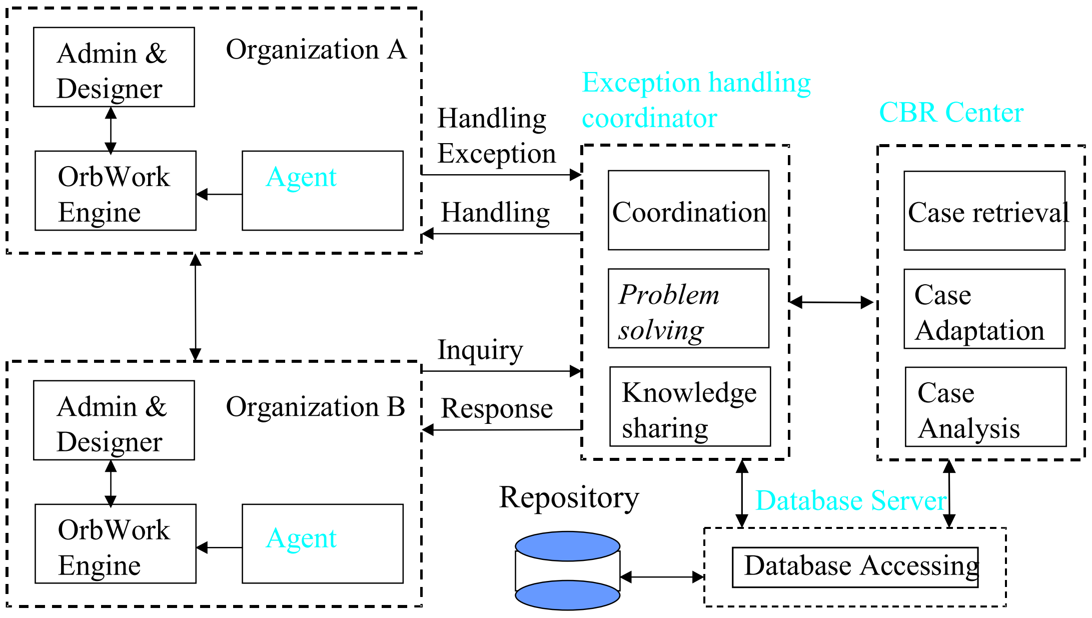


Figure 9.1 Exception handling implementation conceptual architecture

This coordinated exception handling mechanism has been integrated with METEOR OrbWork workflow management system. It is implemented in a distributed manner (see Figure 9.1). A set of exception handling protocols (e.g. Handling, HandleException, Inquiry, and Response) has been created (see Figure 9.1). "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.

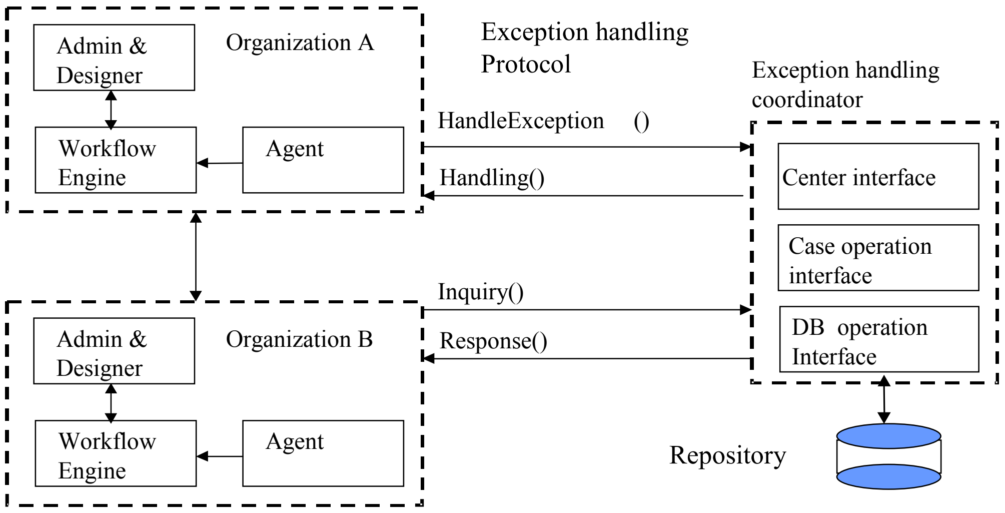
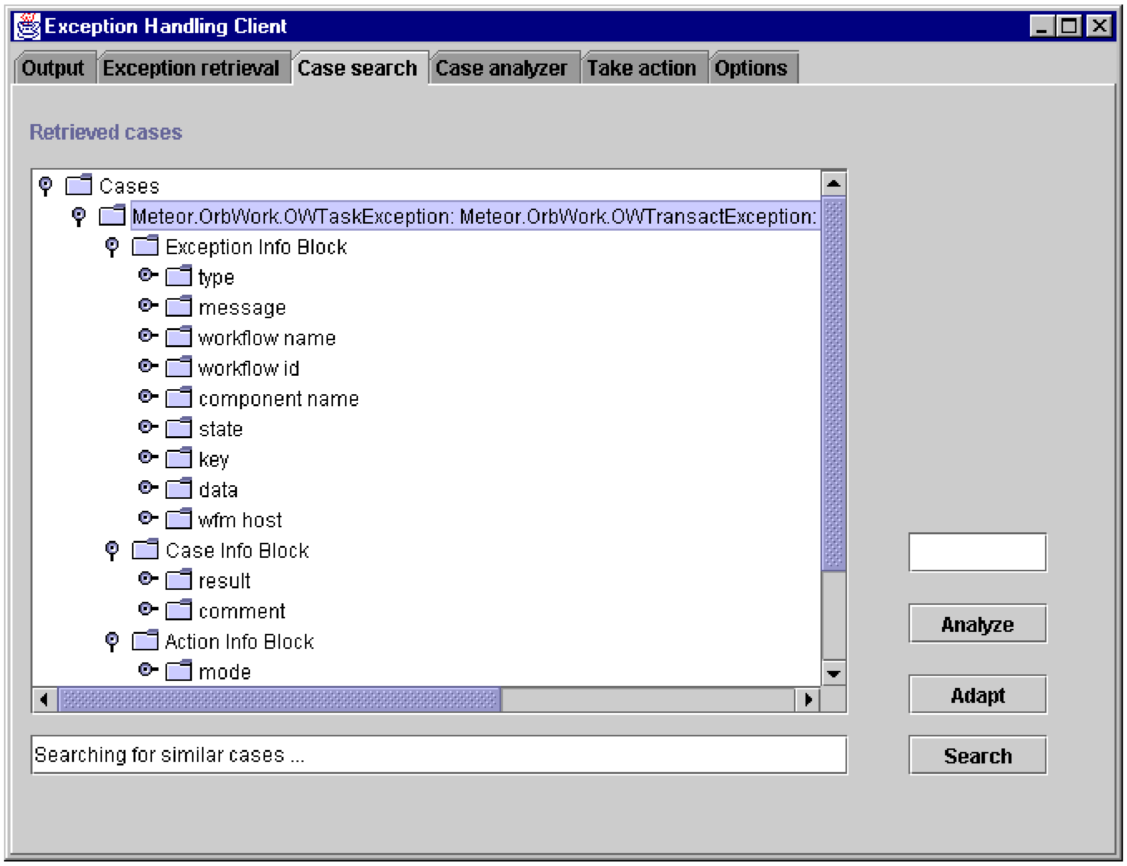


Figure 9.2 Exception handling coordinator implementation architecture

A GUI client (see figure 9.3) 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 9.3), 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.


Exception Handling Client

Output | Exception retrieval | Case search | Case analyzer | Take action | Options

Retrieved cases

```
Cases
  Meteor.OrbWork.OWTaskException: Meteor.OrbWork.OWTransactException:
    Exception Info Block
      type
      message
      workflow name
      workflow id
      component name
      state
      key
      data
      wfm host
    Case Info Block
      result
      comment
    Action Info Block
      mode
```

Searching for similar cases ...

Analyze | Adapt | Search

Figure 9.3 GUI based Exception handling client

In the case search panel (see Figure 9.3) 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 9.3).

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 (see Figure 9.3). The case adaptation process will begin when the “Evolve” button (which is not shown in Figure 9.3) 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 9.3). Administrators can also write back the cases into the case repository.

PROCESS INSTANCE AND DATA BACKUP

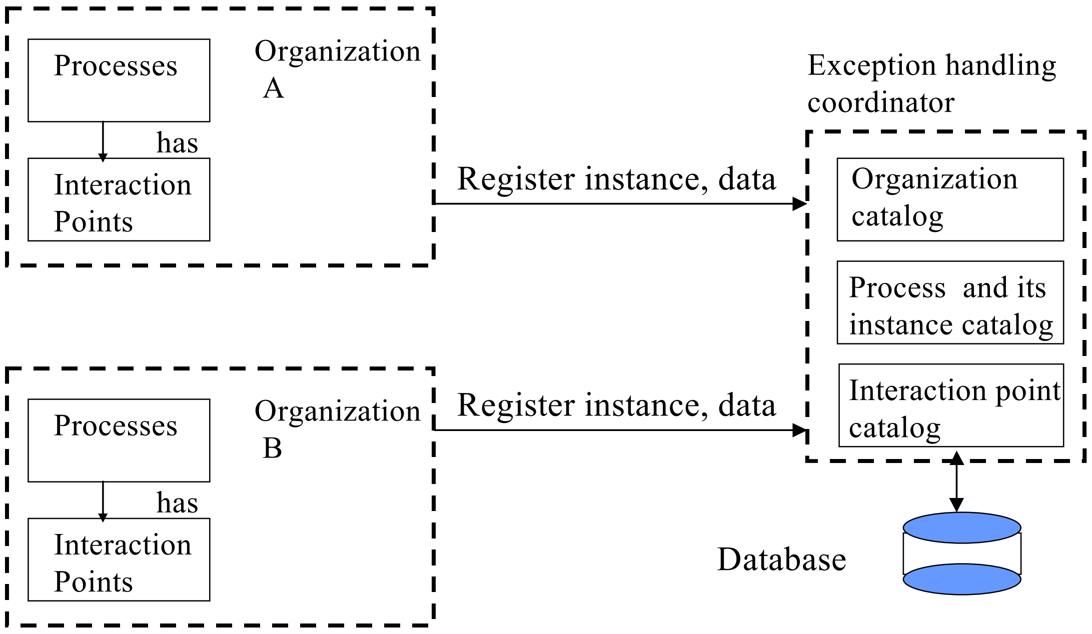


Figure 9.4 Process instance and data registration

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 [Luo 2000] is one of the choices to facilitate recovery process. In this dissertation, 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.

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 9.5.

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.

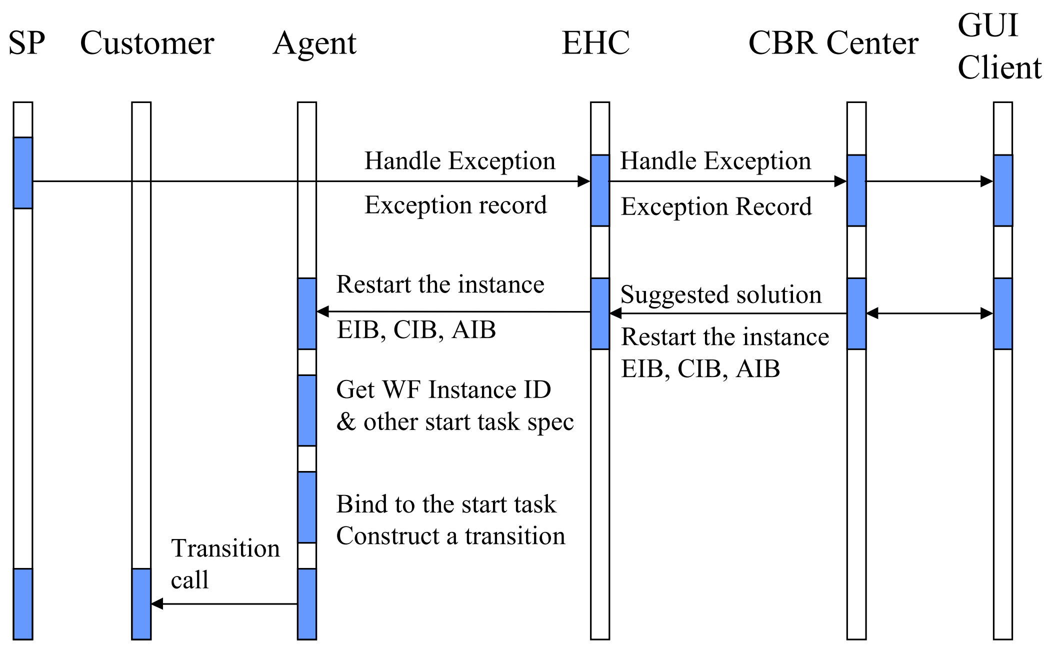


Figure 9.5 Restart a workflow instance

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 9.6. 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.

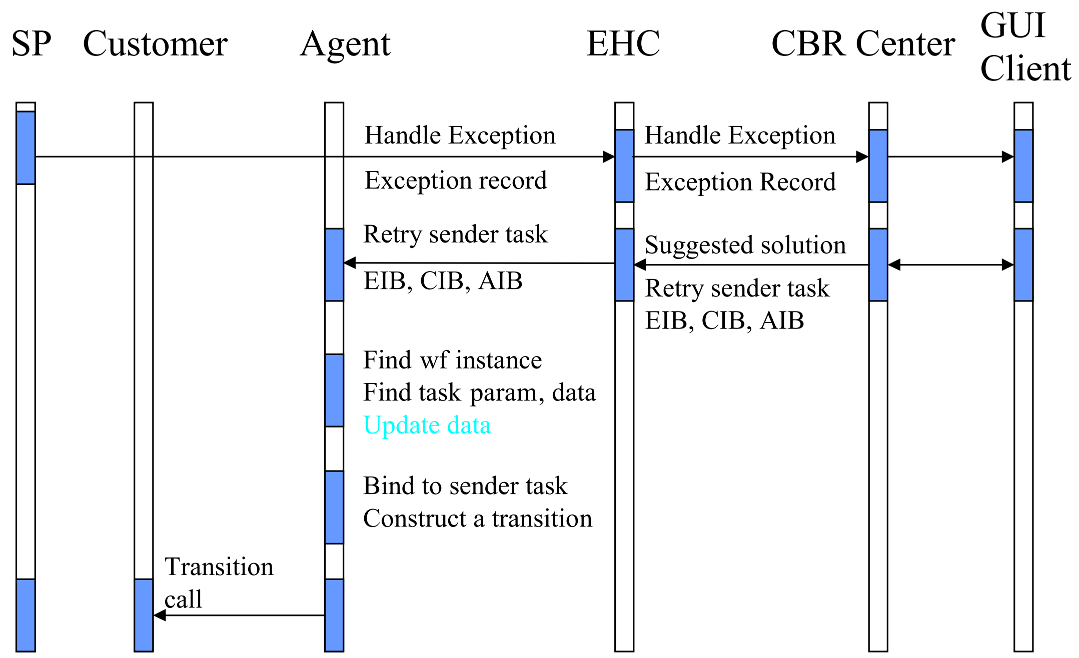


Figure 9.6 Retry and alternative task implementation

EXCEPTION DESIGN AND HANDLING

Workflow application developers can anticipate and provide solutions to deal with certain exceptional situations. To support such user exception handling, we provide tools to allow workflow application designers to define their own exceptions, called user exceptions. In the exception design, it is necessary to provide mappings between exceptions and exception sources (see Figure 9.7). Exception sources include errors, faults, failures, and other exceptional situations. We view constraints as exception sources too. That is, when constraints are not met, exceptions may be raised. By providing such a mapping mechanism, workflow application designers can decide which exceptions should be raised when those abnormal situations occur. Also, workflow application designers can provide mappings from exceptions to exception handlers so that when such exceptions are raised, systems can know which exception handlers are good candidates. The candidates for exception handlers are ignoring, warning, retry, suspend/stop/resume, workflow recovery operations (e.g., backward recovery, forward recovery, alternative tasks, etc.), workflow modifications and evolutions, and other exception masking and propagation operations. Those mappings will be specified in JECA rules [Luo et al 2000]. More information about JECA can be found in the process analysis section in the appendix.

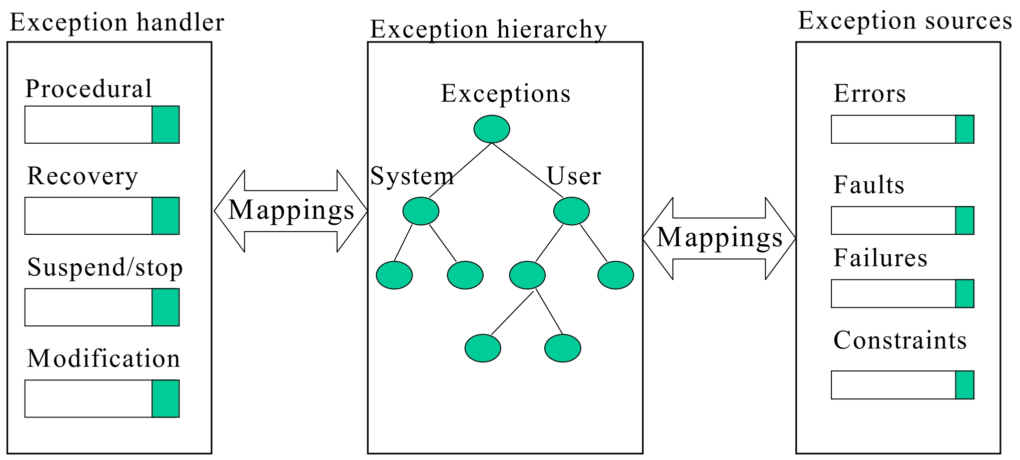
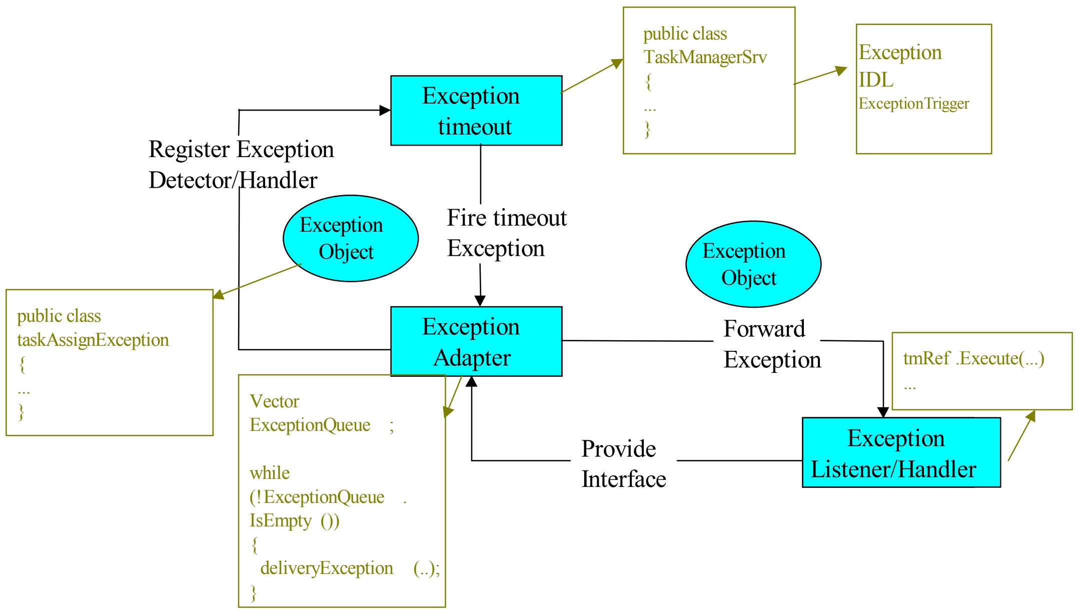


Figure 9.7 Mappings between exceptions, exception sources, and exception handlers



```
public class
TaskManagerSrv
{
...
}
```

```
public class
taskAssignException
{
...
}
```

```
Vector
ExceptionQueue ;

while
(!ExceptionQueue .
IsEmpty ())
{
 deliveryException (..);
}
```

```
tmRef .Execute(...)
...
```

Figure 9.8 User exception handling scenario

The following JECA rule,

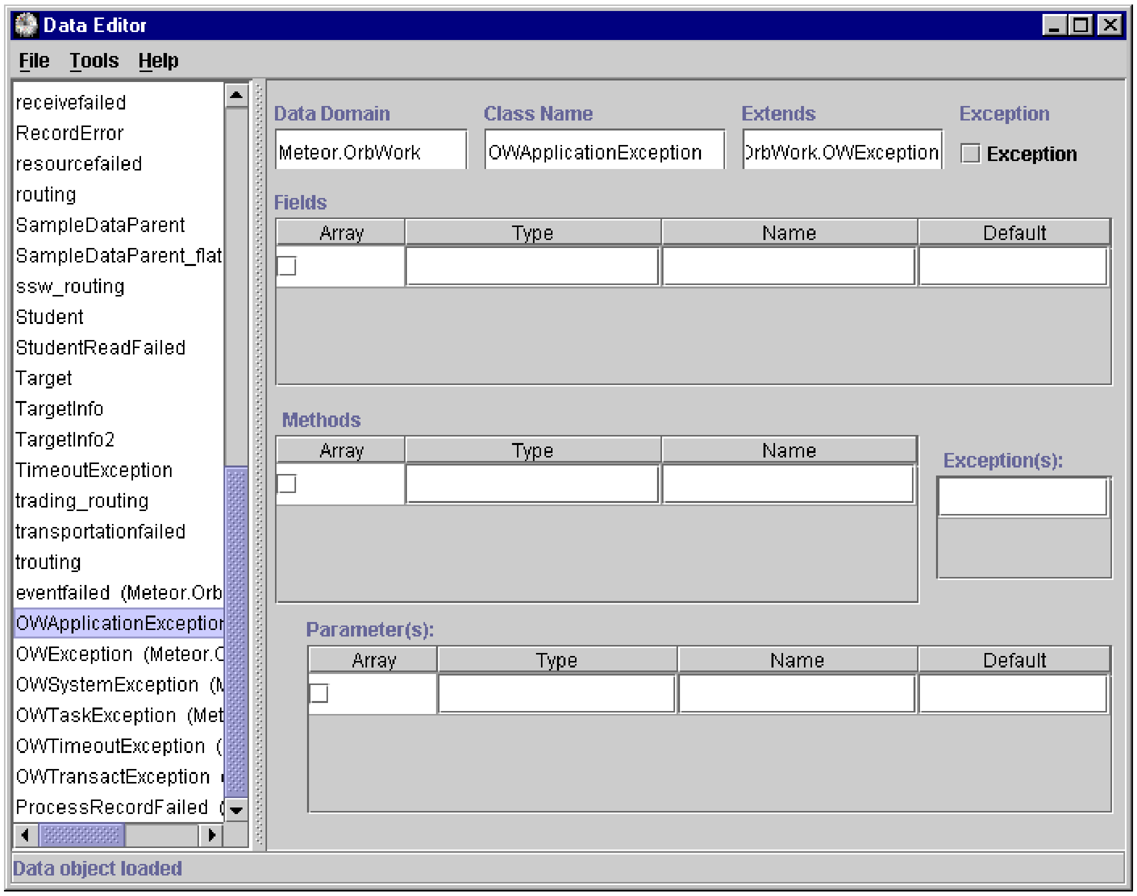
Event: taskAssignException

Condition: agent not available and task priority is urgent

Action: task-reassignment

Justification: newborn infant transport to NICU,

shows a mapping example of a task assignment exception in the infant transport application. In the resource allocation step, if there are no human agents available who can assume the health professional role, then taskAssignException exception will be raised. Since the infant should be transported immediately, another qualified healthcare professional for the healthcare professional role must be found.

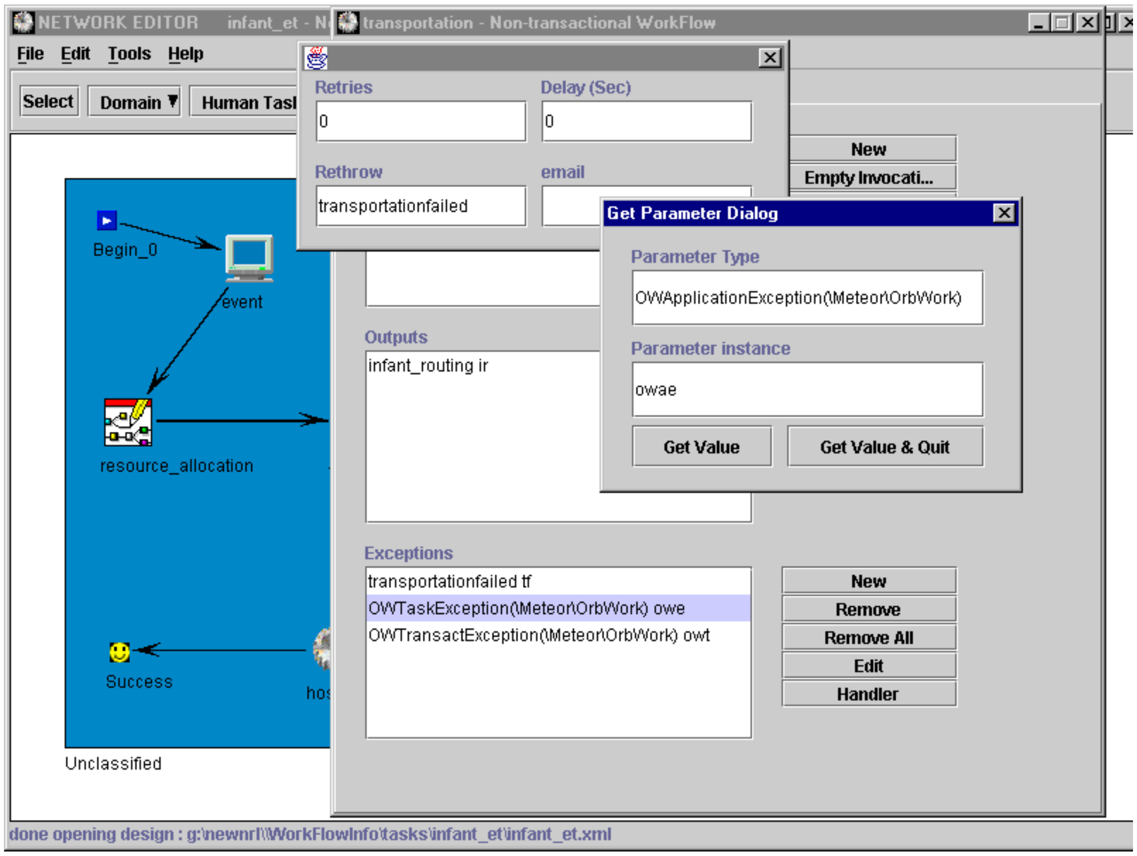


Data Domain	Class Name	Extends	Exception
Meteor.OrbWork	OWApplicationException	)rbWork.OWException	☐ Exception

Data object loaded

Figure 9.9 Exception editor

In Figure 9.8, an example is shown for a user exception-handling scenario that involves taskAssignException exception. This exception will be declared as a member of taskAssignException class. By using the mappings designed, workflow systems will register the exception detector and handler for that taskAssignException exception. When the task assignment exceptional situation as defined by administrators is detected, a taskAssignException is raised. The taskAssignException exception object will be forwarded to the registered exception handler via the exception adapter. The exception adapter in this scenario act as a bridge that de-couples exception detectors and handlers.



Retries: 0
Delay (Sec): 0
Rethrow: transportationfailed
email:

Parameter Type: OWApplicationException(\Meteor\OrbWork)
Parameter instance: owae

infant_routing ir

transportationfailed tf
OWTaskException(\Meteor\OrbWork) owe
OWTransactException(\Meteor\OrbWork) owt

done opening design : g:\newnrl\\WorkFlowInfo\tasks\infant_et\infant_et.xml

Figure 9.10 Exception handling editor

As shown in the Figure 9.9, the exception editor is used to define new exceptions. Exception types can only be used in the workflow applications after they have been defined. At this time, a GUI-based editor is implemented. This exception editor uses the same GUI interface as data editor.

If application developers want to define new exception types, they can choose the "new" item in the "File" menu. Once the "Save" item in the "File" menu is clicked, all the changes will be stored and kept.

Workflow application developers can specify exception-handling schemes by using the GUI tool (see Figure 9.10) developed. As shown in Figure 9.10, there are three types of exception handling schemes that can be specified in this GUI exception handling designer - retry, rethrow, and email.

SYSTEM 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, which holds all the property settings to make the system work.

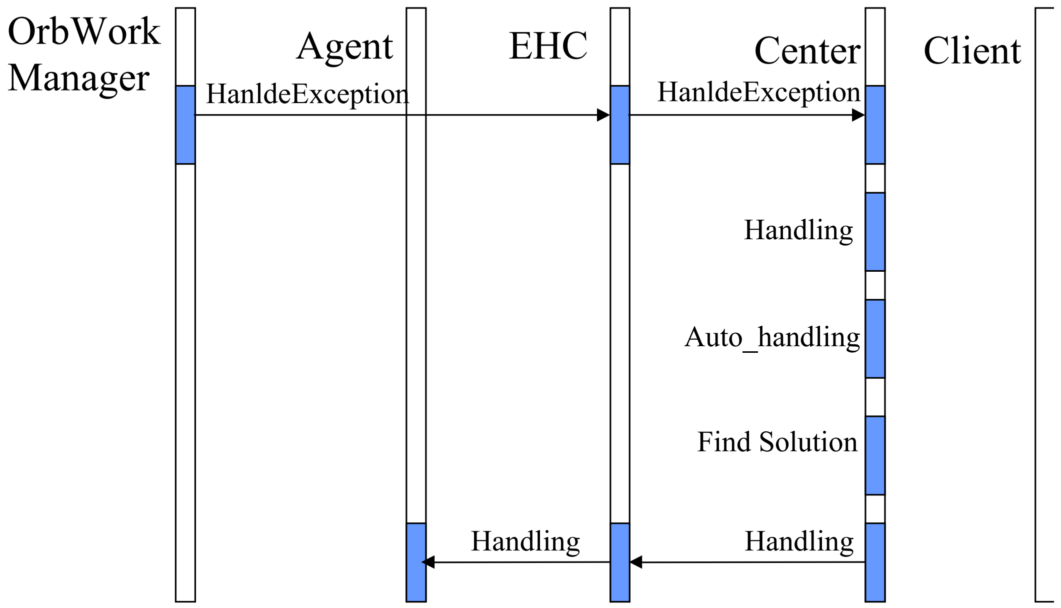


Figure 9.11 Automatic system working mode

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 9.11.

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.

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.

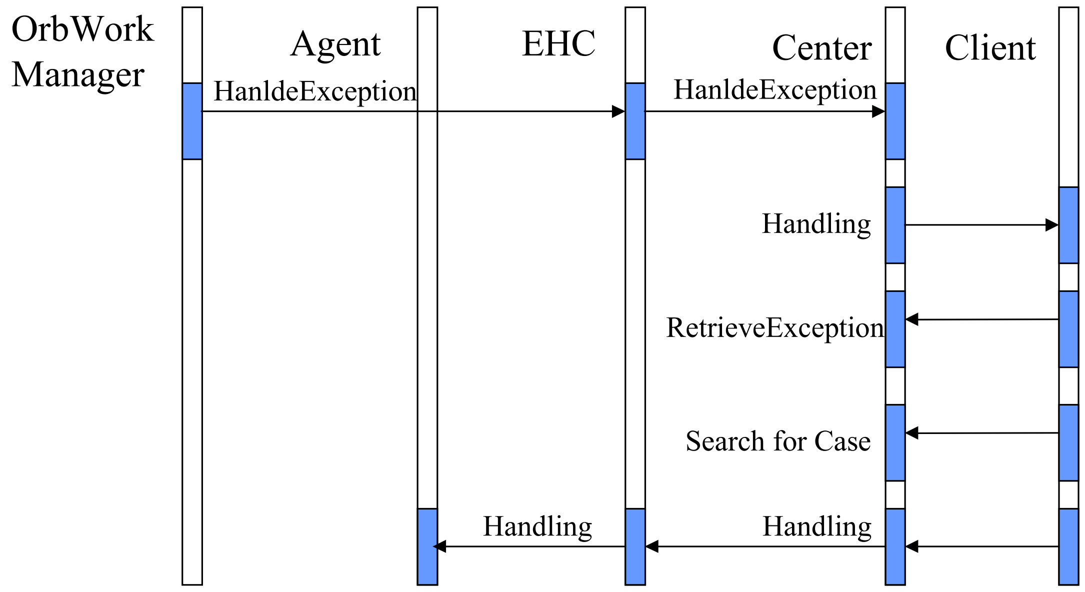


Figure 9.12 Manual handling mode

CHAPTER 10

SYSTEM ANALYSIS AND CONCLUSIONS

In this dissertation, we have proposed a bundled cross-organizational exception-handling scheme. Exception knowledge sharing partly solves the heterogeneity problems we have identified in this paper. The five exception handling modes capture the possible interaction means in cross-organizational exception handling processes to meet various business requirements. Intelligent problem solver helps populate the CPR exception handler. It is based on the techniques of case based reasoning. The essence of CBR is that similar problems can be solved by similar solutions. Does it really work for exception handling? In the following, we will show our evaluation of the applicability of CBR in our exception handling system. The result is encouraging.

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 prior obtained 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. These two experiments are used to test the problem re-occurrence rate, and to understand the distribution of problem re-occurrences. Moreover, we give an analysis about the relationship between encountered problems and their solutions.

In the first experiment, we build the telecommunication workflow application, and execute it as usual. Then we do nothing to the occurring exceptions but record them. There are total 122 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 16 per run.

In the second experiment, we build another application, and execute it as usual. Again, we do nothing to the occurring exceptions but record them. There are a total of 149 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 finding 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.

CBR, which is used to find these kinds of exception patterns, will add great value to the exception handling coordinator.

In the following sections, we will describe our analytical assessment, use-cases assessment, and experimental assessment over the exception handling system. In analytical assessment, we have built a mathematical model to understand what a promising technique CBR is. By using use-cases assessment, we have identified the impact of implementing exception handling in cross-organizational setting upon the ORBWork WfMS. We have also identified the weakness of the ORBWork workflow management system. We have also designed five workflow applications to test various aspects of the exception handling system, which are discussed next.

- Number of process instances that can be run in the modified ORBWork WfMS will be tested in the first experiment.
- We will test the number of process instances that can be run in the modified ORBWork WfMS after there is an exception has been handled by the exception handling coordinator in the second experiment.
- The application exception propagation will be tested in the third experiment.
- An exception will be generated for each process instance. It will be handled by the exception handling coordinator. The number of process instances that can be run in this situation will be tested in the fourth experiment.
- The adaptability of the exception handling system will be tested in the fifth experiment.

ANALYTICAL ASSESSMENT

Here we use a mathematical model based assessment to understand the CBR problem solving capability. To understand whether similar problems have similar solutions, we have created a problem-solution (P-S) matrix. By using this P-S matrix, we obtained four types of relationships between problems and their solutions:

- Similar problem, similar solution. If similar problems can be solved by similar solutions, CBR is the perfect problem solving mechanism.
- Similar problem, different solution. To apply CBR in this situation, an understanding must be achieved about these different solutions. Can these different solutions be reduced to similar solutions? If yes, CBR is applicable. Otherwise, the common intersection among these different solutions must be found. Human involvement intervene is necessary if no intersections can be found.
- Different problem, similar solution. To apply CBR in this situation, an understanding must be achieved about these different problems. That is, CBR should adapt to different situations. Because these difference problems have similar solutions, it is similar to the cases of similar problems having similar solutions.
- Different problem, different solution. In this situation, problems must be solved case by case. The problem solving capability of CBR increases with its case base size. Human intervenes is necessary when the problems are solved at the time of their first occurrences.

We can assume the probability distributions of the problems, and the handling capability of the system. Then we can conduct mathematical calculation to get the insightful ideas about the CBR problem solving capabilities.

Initially,

- The distribution probability of "similar problem, similar solution" is p_{ss}.
- The distribution probability of "similar problem, different solution" is p_{sd}.

- The distribution probability of "different problem, similar solution" is p_{ds}.
- The distribution probability of "different problem, different solution" is p_{dd}.

The sum of these probabilities should be 1. That is, $p_{ss} + p_{sd} + p_{ds} + p_{dd} = 1$

Now we are going to show p_{sd} will not be zero. If we assume p_{sd} can be zero, this means once we solve a problem, it can be used to match other similar problems. This means that using prior gained experiences can solve these problems. However, not all problems are solvable. There exist some problems that are not solvable according to time and space constraints. There are problems that can nt be solved even by experts. Thus, it is not possible to get a problem-solution pair for all the problems. So p_{sd} will be larger than 0.

The best chance for us is how we can transfer problems from other categories to the category of "similar problem, similar solution". We can assume the transition probability from each category to "similar problem, similar solution". The reason is once a problem is encountered by the system, it is known to the system. Later it is possible that a similar problem may be encountered again.

The transition probability of from "similar problem, similar solution" to "similar problem, similar solution" is 1. This is because they are in that category.

The transition probability of from "similar problem, different solution" to "similar problem, similar solution" is $p_{sd\text{-}ss}$. Usually it is always 0. This is because for a recurring similar problem, always a different solution from the one contained in the prior encountered case is needed.

The transition probability of from "different problem, similar solution" to "similar problem, similar solution" is $p_{ds\text{-}ss}$. This transition is possible because once a problem is met, it become a base for the similarity match. If this kind of problems always has solutions, $p_{ds\text{-}ss}$ is larger than 0. The transition probability of "different problem, similar solution" to "similar problem, different solution" is $p_{ds\text{-}sd}$. The sum of the transition probabilities of $p_{ds\text{-}ss}$ and $p_{ds\text{-}sd}$ is 1. That is, $p_{ds\text{-}ss} + p_{ds\text{-}sd} = 1$. Usually $p_{ds\text{-}ss} > p_{ds\text{-}sd}$ holds.

Similarly, the transition probability of "different problem, different solution" to "similar problem, similar solution" is $p_{dd\text{-}ss}$. This transition is possible because once a problem is met, it become a base for the similarity match. If this kind of problems always have similar solutions, the $p_{dd\text{-}ss}$ is larger than 0. The transition probability of "different problem, different solution" to "similar problem, different solution" is $p_{dd\text{-}sd}$. The sum of the transition probabilities of $p_{ds\text{-}ss}$ and $p_{ds\text{-}sd}$ is 1. That is, $p_{ds\text{-}ss} + p_{ds\text{-}ss} = 1$. Usually $p_{dd\text{-}ss} < p_{dd\text{-}sd}$ holds.

Now it is possible to calculate the evolving distribution probability of "similar problem, similar solution" and "similar problem, different solution". Once a new problem is encountered, the distribution probability of it belonging to "similar problem, similar solution" is:

- p_{ss} if the encountered problem belongs to "similar problem, similar solution".
- 0 if the encountered problem belongs to "similar problem, different solution". This is because there is no transition from "similar problem, different solution" to "similar problem, similar solution".
- $(p_{ds\text{-}ss} * p_{ds})$ if the encountered problem belongs to "different problem, similar solution". This is because the distribution probability of "different problem, similar situation" is p_{ds}. Considering the transition probability of $p_{ds\text{-}ss}$, the final probability of the encountered finally belonging to "similar problem, similar solution" is $(p_{ds\text{-}ss} * p_{ds})$.
- $(p_{dd\text{-}ss} * p_{dd})$ if the encountered problem belongs to "different problem, different solution". This is because the distribution probability of "different problem, similar situation" is p_{ds}. Considering the transition probability of

$p_{dd\text{-}ss}$, the final probability of the encountered finally belonging to "similar problem, similar solution" is ($p_{dd\text{-}ss}$ * p_{dd}).

So once a new problem is encountered, the new distribution probability of "similar problem, similar solution" is (p_{ss} * 1) + (p_{sd} * 0) + ($p_{ds\text{-}ss}$ * p_{ds}) + ($p_{dd\text{-}ss}$ * p_{dd}). Similarly we can obtain the new distribution probability of "similar problem, different solution" when a new problem is encountered. It is (p_{sd} * 1) + (p_{ss} * 0) + ($p_{ds\text{-}sd}$ * p_{ds}) + ($p_{dd\text{-}sd}$ * p_{dd}).

The evolving rate of the distribution probability of "similar problem, similar situation" is ($p_{ds\text{-}ss}$ * p_{ds}) + ($p_{dd\text{-}ss}$ * p_{dd}). The evolving rate of the distribution probability of "similar problem, different situation" is ($p_{ds\text{-}sd}$ * p_{ds}) + ($p_{dd\text{-}sd}$ * p_{dd}).

From this analysis, we can draw the following conclusions:

- CBR cannot be used to solve all the problems.
- The best CBR problem solving capability is 1- p_{sd}. p_{sd} is pre-determined by the capability of both human and computing systems.
- ($p_{ds\text{-}ss}$ * p_{ds}) and ($p_{dd\text{-}ss}$ * p_{ds}) denote the possible capability gain for CBR. Initially the sum of (p_{ds} + p_{dd}) is 1, which means no cases exist in the case repository.
- For ($p_{ds\text{-}ss}$ * p_{ds}) type problems, CBR system needs strong adaptation capability to the problems.
- For ($p_{ds\text{-}ss}$ * p_{ds}) type problems, CBR system needs strong adaptation capability to the both problems and solutions.

When we are applying this CBR based exception handling system to handling exception in various applications or domains, we can use this P-S matrix to analyze the effectiveness of this exception handling system. The effectiveness or applicability is determined by the probability distribution, i.e. the above parameters such as p_{ss}, p_{ds} ,

p_{sd}, p_{dd} , etc. in the P-S matrix. This matrix should be able to obtain through statistical analysis or through simulation.

USE-CASES ASSESSMENT

Use-cases technique [Kulak and Guiney 2000] is usually seen as a requirement analysis tool for describing and verifying system functionality. Here we use it to access system quality attributes. This technique is directly dependent on the profile defined for the quality attributes that are to be assessed. Its effectiveness is largely dependent on the representatives of the use case scenarios. The reason that we use use-cases is that the system architectural design will be optimized for this set of use-cases. Since we don't have a similar cross-organization systems to compare, this use-case based assessment is used for comparing ORBWork with cross-organizational exception handling capability and the ORBWork without exception handling capability.

By using use-case techniques, we have conducted impact analysis, i.e., the numbers of lines of code affected to the ORBWork code. Since performance needs quantity-based assessment, we will use experimental assessment to evaluate the system properties. Here we will use five scenarios to evaluate ORBWork system's capability in supporting exception handling in cross-organizational settings.

IMMEDIATE MODE SCENARIO

We re-draw the scenario we have for describing immediate exception handling mode in the coordinated exception-handling chapter. As shown in the Figure 10.1, SPs route their bandwidth change requests to L3 due to customer needs growth. They will immediately get an exception in case L3 determines that requested bandwidth range is not existing. This exception is generated by database system because the database is happy with the entry contained in the request - database statement has no results.

We have tried to implement this mode of exception handling. We have found that this type of exception handling mode is most suitable for those well-known exceptions with well-known exception handlers. Thus, once we have identified those well-known exceptions with well-known handlers, it is possible to put them directly into the system code section by using try-catch blocks. This mode works in tight-coupled systems. However, this also makes it inflexible.

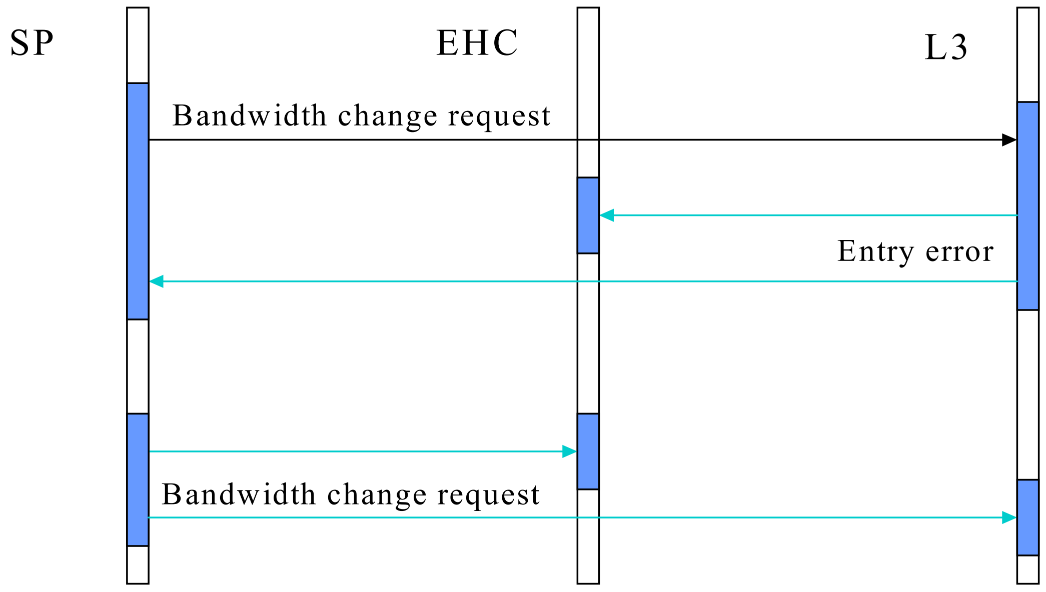


Figure 10.1 Immediate mode of cross-organizational exception handling

The rethrow handling scheme that can be designed through workflow designer is working in immediate exception handling mode. Designer does not support other local exception handlers at this time except retry and email.

DEFERRED MODE SCENARIO

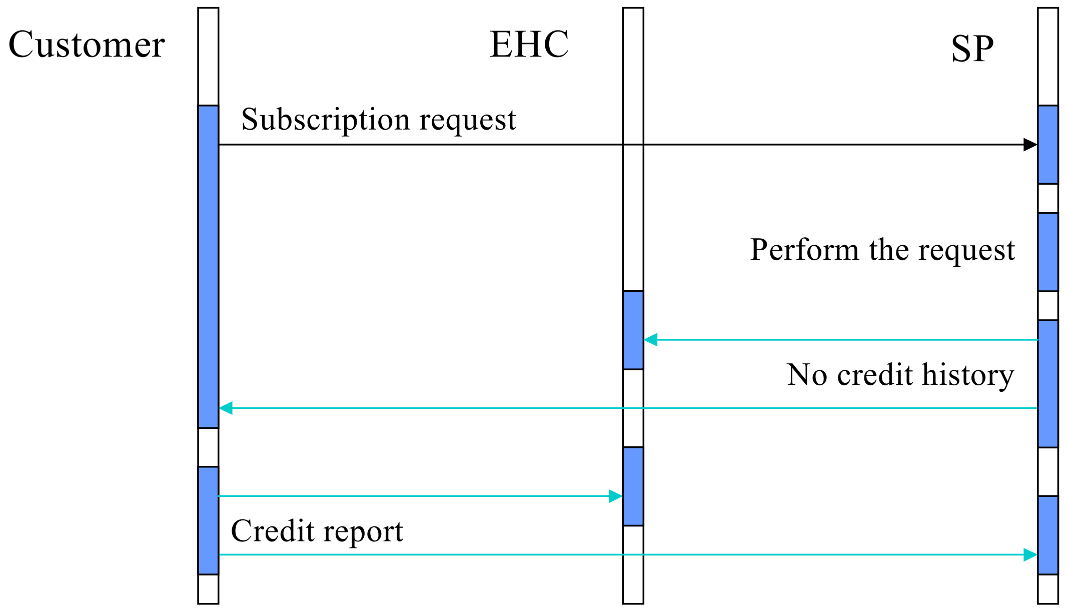


Figure 10.2 Deferred mode of cross-organizational exception handling

We re-draw the scenario we have for describing deferred exception handling mode in the coordinated exception-handling chapter. As shown in the Figure 10.2, SPs route its customers' bandwidth requests to L3. L3 will not raise exception immediately to these SPs when it determines that not enough credit information is available along with the request. Instead, L3 will fulfill the request, and later raise the exception to SPs through the same two-way interaction point.

We have tried to implement this type of exception handling mode. But clearly the ORBWork workflow management system does not support this type of coordination. A coordination scheme similar to nested sub-workflow instance interoperability needs to be supported.

DE-COUPLED MODE SCENARIO

We re-draw the scenario we have for describing de-coupled exception handling mode in the coordinated exception-handling chapter. As shown in the Figure 10.3, 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 514KBPS channel is not available, it needs to raise this exception to SPs. Since there is no other interaction points between them, L3 needs to raise it to SPs through the exception handling coordinator.

We have implemented this type of exception handling mode. We have found ORBWork workflow management supports it quite well, following several implementation-specific changes to the ORBWork. In the current ORBWork implementation, in case of task failure or abort, along in the exception propagation route, all the workflow instances will be removed from the task schedulers. So we need to register and make a reference copy of these instances. During the de-coupled exception handling, these instances will be used.

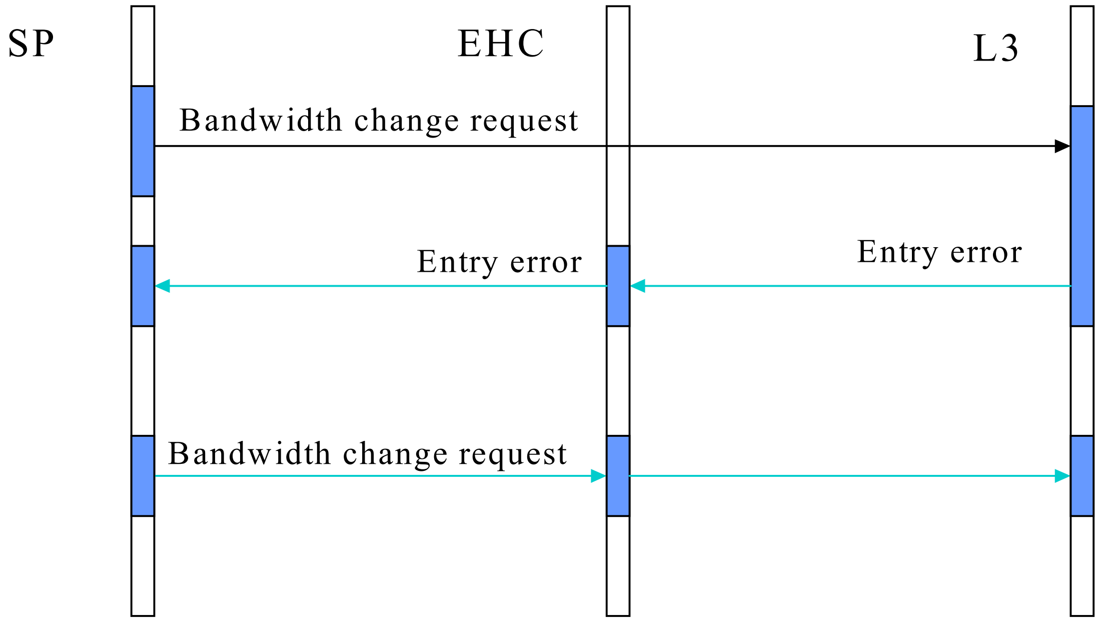


Figure 10.3 De-coupled mode of cross-organizational exception handling

FREE MODE SCENARIO

We re-draw the scenario we have for describing immediate exception handling mode in the coordinated exception-handling chapter. In the Figure 10.4, SPs route its customers' bandwidth requests to L3. L3 will not raise exception SPs if it determines that too many (not realistic) channels are typed in along with the request. Instead, L3 will fulfill the request partially, and later raise the exception to SPs through another interaction point determined by the exception handling coordinator.

Currently ORBWork implementation does not support this type of exception handling mode yet. However, with the implementation of the dynamic changes, this type of coordination can be supported in the future ORBWork implementations. One of the key changes is to add an exception object to the normal flow. There are several process analyses going on when an exception object is added, such as termination, reach-ability, etc. Another key change to be made is to detect that there is an exception object in the normal flow data.

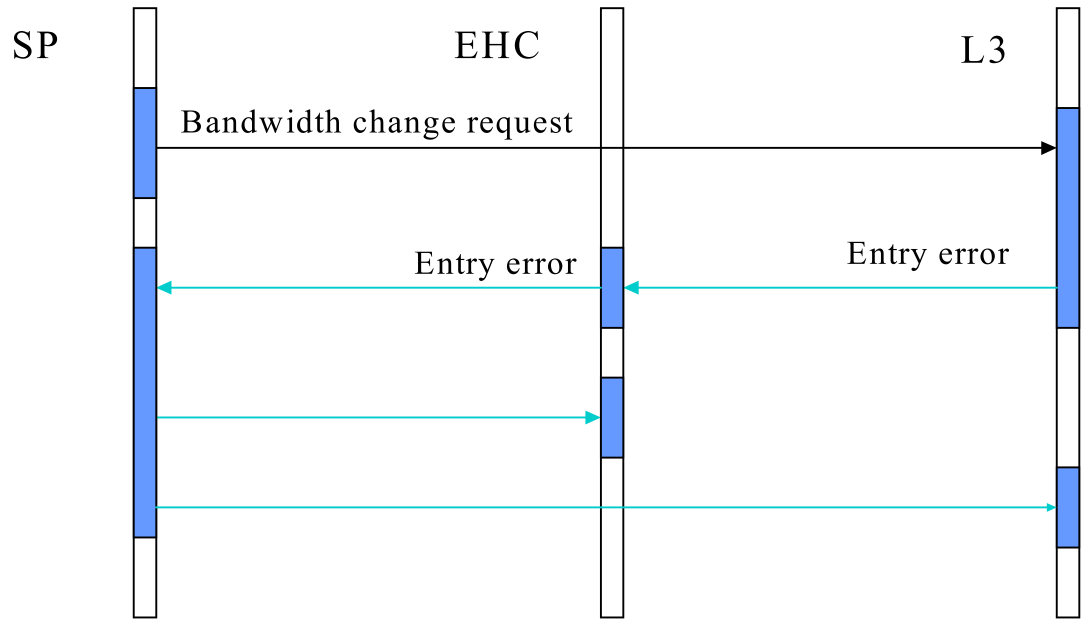


Figure 10.4 Free mode of cross-organizational exception handling

Close Mode scenario

We re-draw the scenario we have for describing immediate exception handling mode in the coordinated exception-handling chapter. In the Figure 10.5, SPs determines the service quality provided by L3 is not satisfactory, since 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.

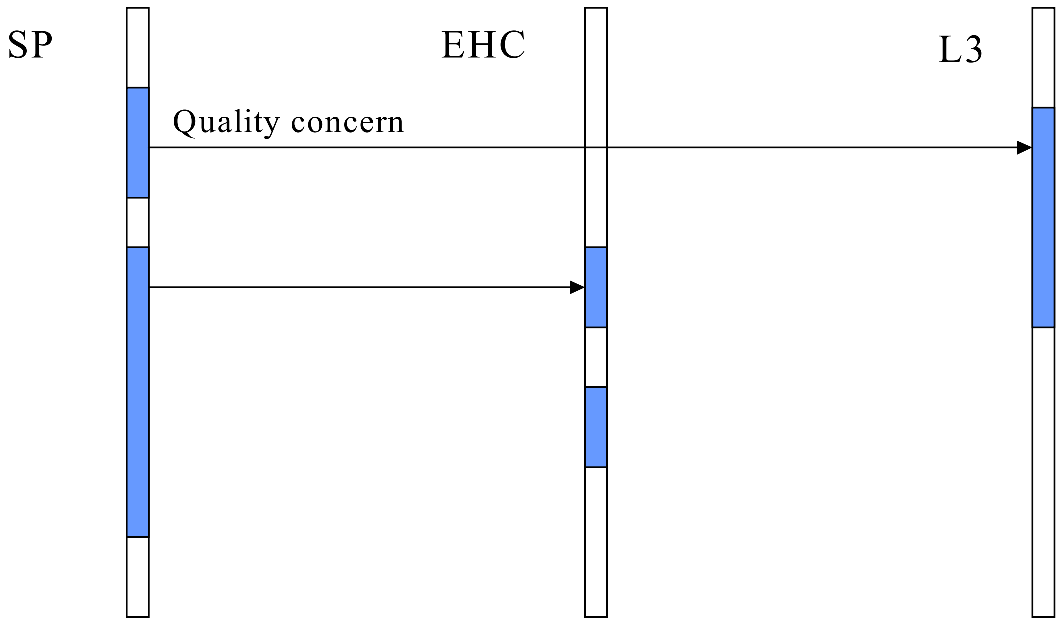


Figure 10.5 Close mode of cross-organizational exception handling

Current ORBWork and exception handling system implementation support this type of coordination. The challenge here after the exception is recorded, how much benefit it can provide besides statistical analysis. This is dependent on the system users.

EXPERIMENTAL ASSESSMENT

To get more quantitative assessment, we have conducted experimental assessment over the exception handling system. This is made possible because we have implemented all the components of the system architecture (see Figure 9.1) and we have all the context of the system. We have designed five different applications to assess whether the system functions correctly or not in different circumstances.

Experimental assessment complements the use-cases based approach in that experimental assessment is particularly useful for evaluating operational quality attributes, such as performance of exception handling by actually executing the system implementation, whereas use-cases are more suited for evaluating development quality attributes, such as flexibility.

EXPERIMENT DESIGN

There are three types of exceptions, infrastructure exception, workflow system exception, and application exception. However, for system exceptions, if they are caused by implementation error, then there is a danger of losing control in the experiment. The exceptions may get lost. Because of wrong implementation, the system can not work correctly. So instead of testing system exceptions, we have designed experiments to test whether the system is running correctly. If the system is tested to be correct, we go on testing infrastructure exceptions and application exceptions.

In virtually all experiments, some considerations have to be given to the number of repeat tests for the ORBWork workflow management system. A balance has to be struck between the marginal cost per experimental test and the increase in precision achieved per additional test. Except in rare instances where these costs can both be quantified, a decision on the size of experiment is largely a matter of judgement. Some of the more formal approaches to determining the size of experiment usually have spurious precision. Though it is very desirable to make an advance approximate calculation of the precision likely to be achieved, it is usually hard to obtain. We set the maximum size (around 100 workflow instances) of the experiment by our observations. We use two Dell machines. One is running at 466Mhz with 128MB main memory. The other is running at 400Mhz with 256MB main memory. Both machines have Microsoft Window NT operating system (Version 4.0) installed.

Experiment - Service subscription application

Application description

Currently, most large DSL companies sell directly to large business or through telecommunication resellers, such AT&T, and indirectly to residential customers and small business users via ISPs. As shown in Figure 10.8, ender users (residential users and small business users) subscribe to ISP resellers. These resellers then need to have the order pre-qualified by Digital Local Exchange Carrier (DLEC). These orders again need to be routed to Incumbent Local Exchange Carriers (ILEC). Once they are confirmed and delivery date has been determined, the subscribers will be notified.

Experiment purpose

This experiment is used to test the effects caused by the changes we have made to the ORBWork system. As mentioned in the use-cases assessment, some changes were made to support handling exceptions in cross-organizational settings. In this experiment, we have tested how many instances can potentially run in the system. Our target is to set to one hundred workflow instances running in the system.

Exception generation

We do not generate exceptions in this experiment. The ORBWork system is running as normal, i.e., no exceptions are generated on purpose.

Result evaluation

We were able to generate more than one hundred workflow instances at almost the same time. This is because we don't have parallel machine, the instances were actually generated one by one. Because the instance completion time is much longer than the instance generation time, we consider they are generated at almost the same time. The system was working fine. However the machine on which the test was conducted was running very slow. This is partly because that the machine is not fast enough (the main frequency is about 466MHZ) and the main memory is not big enough (about 128MB). Besides this, we have found that for each executing task, there is a virtual JAVA machine running for it. So the throughput of the systems can be evaluated against the number of the tasks in the workflow.

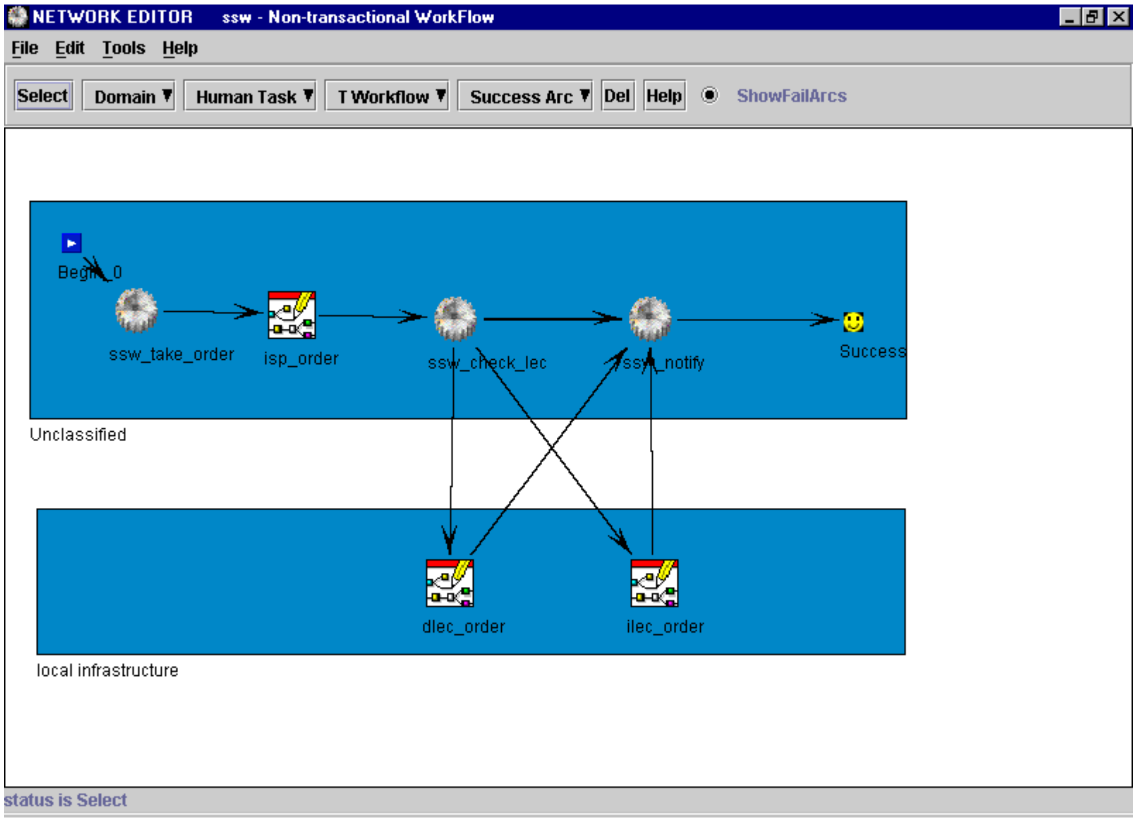


Figure 10.6 Service subscription workflow

EXPERIMENT - DSL APPLICATION

APPLICATION DESCRIPTION

Line sharing and collocation rules are the outgrowths of the Telecommunication Act of 1996. It is designed to promote competition in the telecommunication market. Because of this Act, major DSL providers plan to begin simpler installations of their high speed data services, courtesy of a government mandate requiring local phone companies to share their lines with DSL providers. As shown in Figure 10.8, end users (residential users and small business users) subscribe to ISP resellers. These resellers then only need to have the order pre-qualified by DLEC. Once they are confirmed and their existing phone line has bee upgraded, the subscribers will be notified.

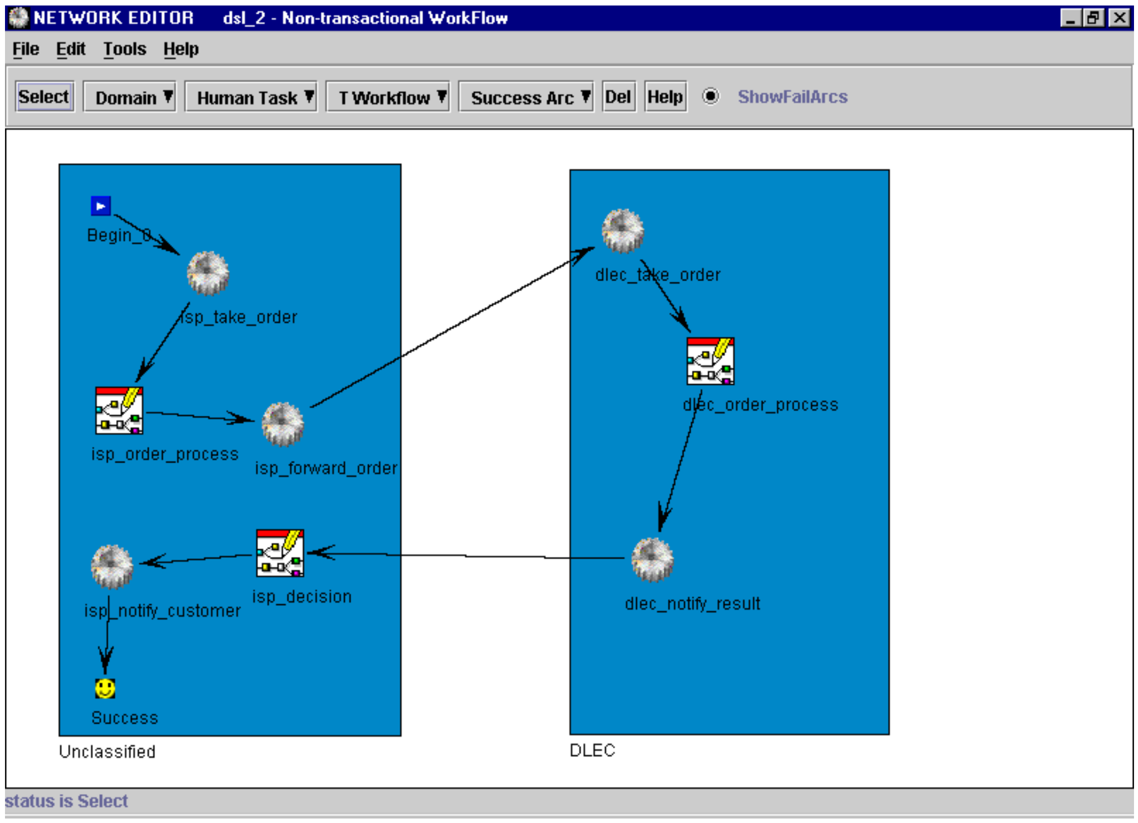


Figure 10.8 dsl subscription workflow

EXPERIMENT PURPOSE

This is the experiment used to test the correct execution of ORBWork system after failures have been remedied.

EXCEPTION GENERATION

To generate an exception that can be handled by the ORBWork system, we have tried several approaches. Finally we found that if we supply an incorrect or inappropriate statement to the database systems (we are using Mini-SQL database server), an exception will be thrown.

Result evaluation

After we had generated the exception and handled it, we were able to generate about one hundred workflow instances almost at the same time. This result is comparable to the experiment conducted using the service subscription workflow.

Application description

We re-draw the infant transportation application in Figure 10.9. This infant transportation application involves the transportation of a very low birth weight infant from a rural hospital to the Neonatal Intensive Care Unit (NICU) at the Medical College of Georgia (MCG). More detailed information about application is given in Chapter 1.

Experiment purpose

In this experiment, we plan to test application exception handling. That is, application designers will design application exceptions using the exception editor. The application designers are responsible to provide mappings between the application exceptions and workflow system exceptions. Designers are also responsible to supply handling schemes. Currently there are three schemes supported by exception designer: retry, re-throw, and email. To test the exception handing system, we use the exception-handling designer to design exception re-throw scheme so a re-thrown exception can be propagated to the exception handling system. Re-throw in essence is a mapping among exceptions. Application designers can use re-throw to provide mappings from one type of exception to another type of exception.

Experiment - Infant transportation application

Exception generation

To generate exceptions, we removed a task from the system. When that removed task is asked to provide services, a null pointer exception will be generated because the requested task is not in the system. The application designers are responsible to provide mappings between their application exceptions and the system exceptions generated.

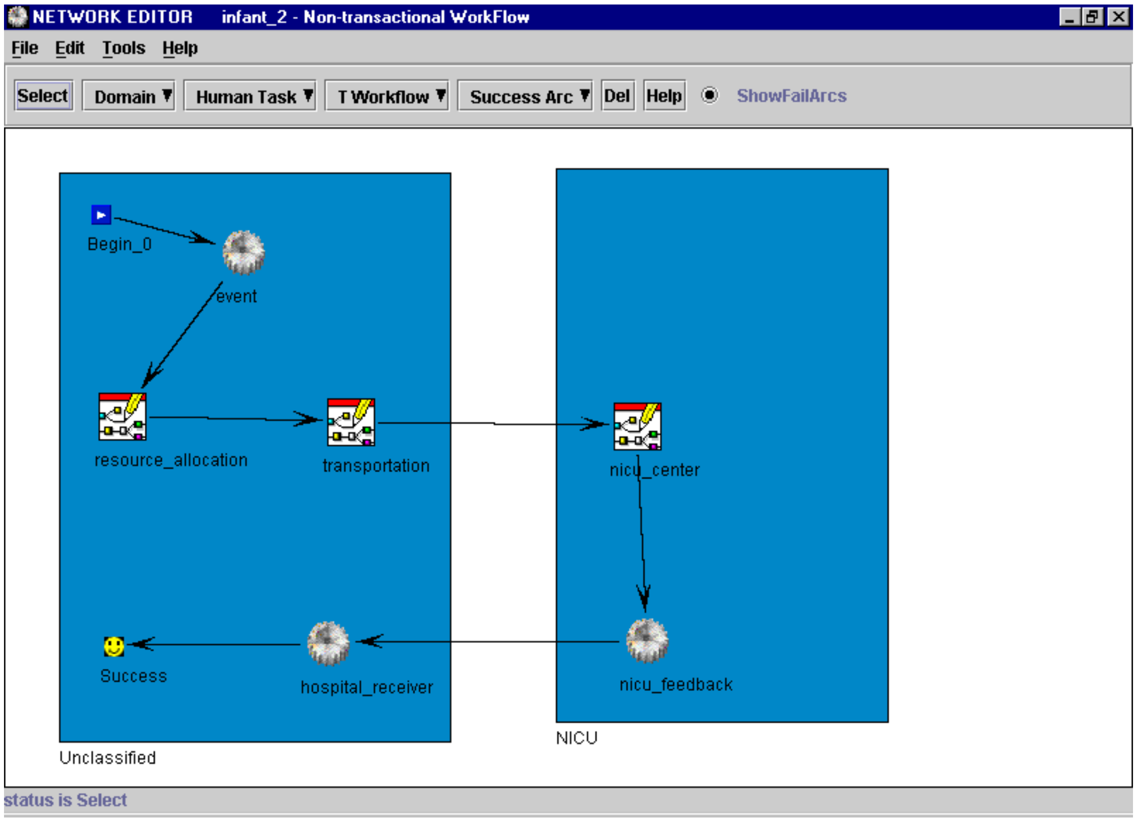


Figure 10.9 Infant transportation workflow

RESULT EVALUATION

We have designed about eight-application exception to infrastructure mappings. We have obtained them all. This means the re-throw exception-handling scheme is working. The workflow designer can choose to map a workflow system exception to an application exception that may be more meaningful to end-users.

EXPERIMENT - LEVEL 3 TELECOMMUNICATION INFRASTRUCTURE PROVIDER APPLICATION

APPLICATION DESCRIPTION

We re-draw the level 3-telecommunication application in 10.10. In this application, 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. More detailed information about this application can be found in Chapter 1.

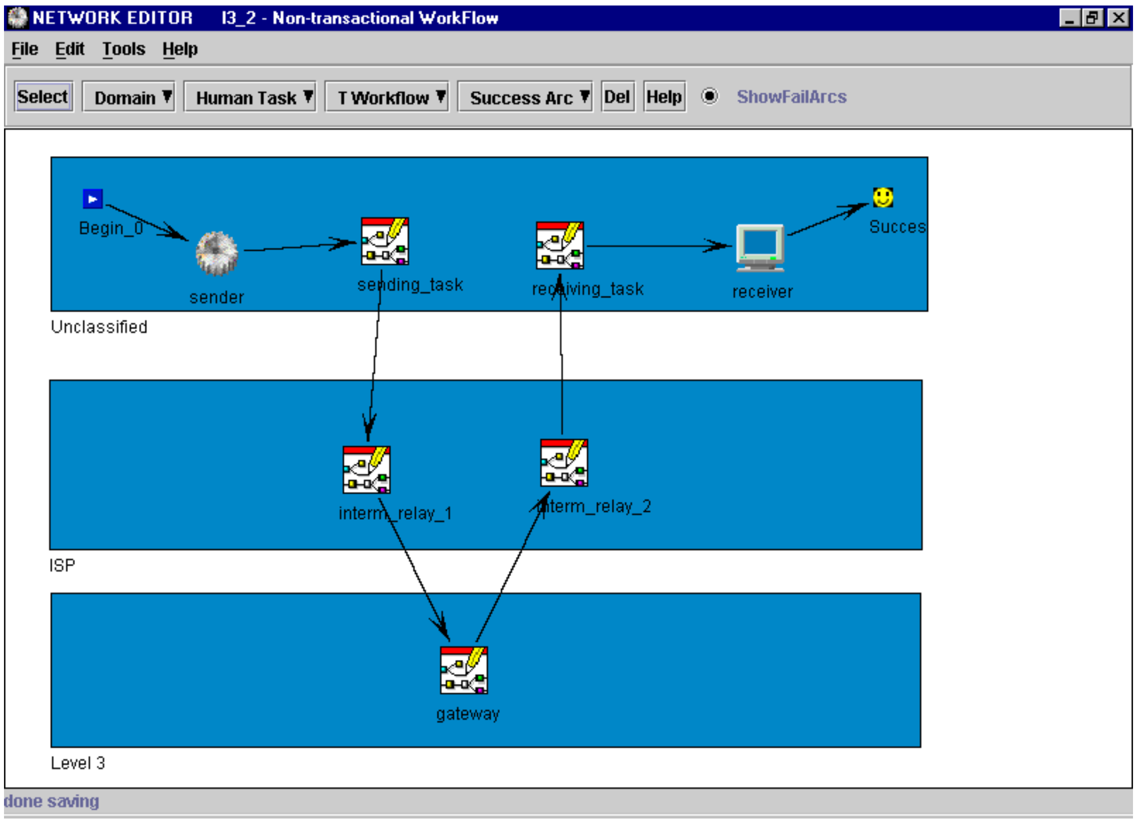


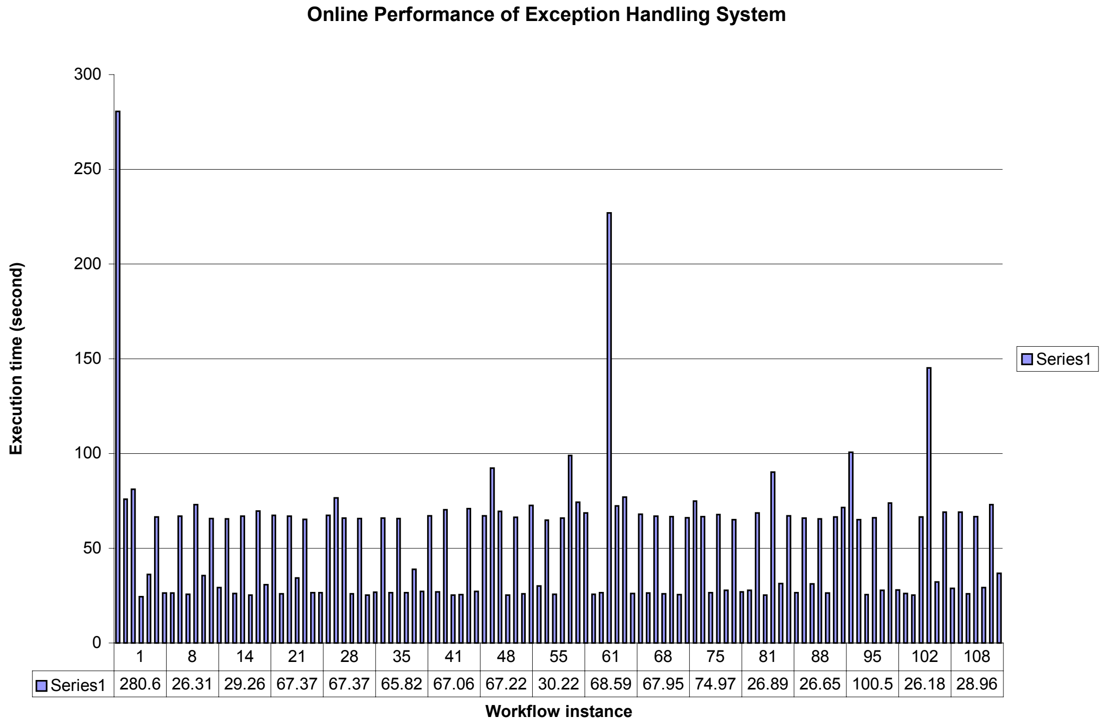
Figure 10.10 Level 3-telecommunication infrastructure provider application workflow

EXPERIMENT PURPOSE

In this experiment, 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. This exception will then be propagated to the exception handling system. This means, the exception handling system is participating in each workflow instance execution.

Exception generation

To generate an exception that can be handled by the ORBWork system, we have tried several approaches. We found that if we supply an incorrect or inappropriate statement to the database systems (we are using Mini-SQL database server), an exception will be thrown.

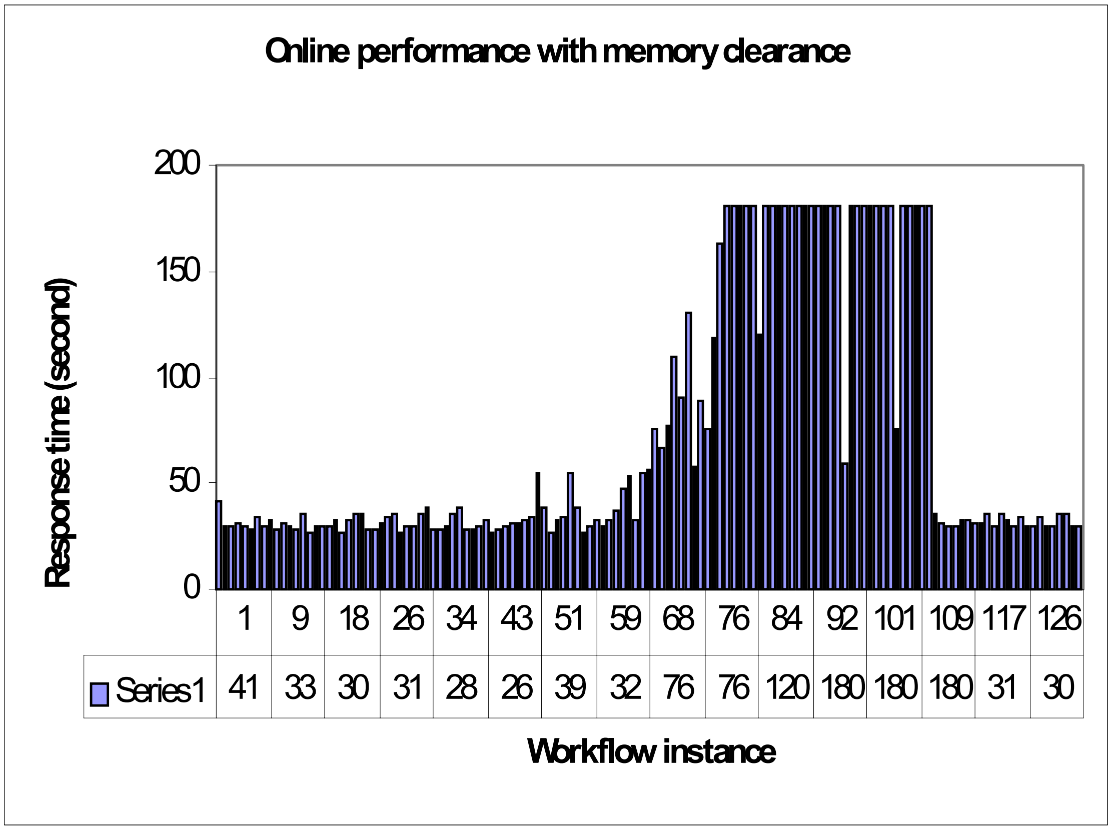


	1	8	14	21	28	35	41	48	55	61	68	75	81	88	95	102	108
▨ Series1	280.6	26.31	29.26	67.37	67.37	65.82	67.06	67.22	30.22	68.59	67.95	74.97	26.89	26.65	100.5	26.18	28.96

Figure 10.11 Execution time of the handling exceptions

Result evaluation

In this test, we were able to generate about 108 workflow instances with concurrently running exception handling system. 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. Compared with the ORBWork system without exception handling system currently running for each workflow instance, the system in this experiment is slower. We have found the reason through the viewing the memory usage under NT's task manager. The main memory is one of the factors. We can see that the execution time for several workflow instances is quite large. The reason is because when the system is first started, much time needs to be spent on the system preparation, such as loading up object.



	1	9	18	26	34	43	51	59	68	76	84	92	101	109	117	126
▨ Series1	41	33	30	31	28	26	39	32	76	76	120	180	180	180	31	30

Figure 10.12 System performance with cooperating exception handling

To improve the system performance, we have conducted another experiment to test the exception handling system with memory clearance. As discussed above, memory is one of the major factors affecting the performance of the exception handling system, if we can collect unused memory spaces and release them, then we can improve the system performance. However, there is no implicit memory free means in JAVA like in C/C++. So we have decided to use multiple exception handling coordinators with each operating at one time. When one exception handling coordinator reaches the performance threshold, another exception handling coordinator will be used. This solves the memory release problem. We call this cooperating exception handling. The performance is shown in Figure 10.12.

From the figure 10.12, when one of the cooperating exception handling coordinator reaches the performance threshold which can be configured, another coordinator will be active and all the exceptions will be routed to this new coordinator to reduce response time. The threshold should be based on historical data or simulations. In the Figure 10.12, it is quite clear that the threshold should be set to 50 seconds. This threshold represents the best performance of current exception handling coordinator implementation. Further improvements can be achieved by reducing memory usage by the implementation, and using more advanced case operation algorithms in the implementation.

EXPERIMENT - FUTURE AND STOCK TRADING APPLICATION

APPLICATION DESCRIPTION

In 1980s, the Securities & Exchange Commission (SEC) was particularly concerned that NASDAQ market makers were offering better quotes via private trading systems or electronic communications networks (ECN), which cater mainly to institutional investors, than they were to the general public. To fulfill their disclosure obligations the ECNs are required to make their best quotes publicly available on the NASDAQ. This rule change paved the way for the ECNs to make all the prices they carried visible to the public, making the ECNs attractive alternatives for investors. Since the SEC issued rules to require market makers to publicly display their best prices for each issue, the pricing has been becoming more and more transparent. At the same time, with the fast development of computer and networking technology, computerized trading has been a reality. The volatile market environment offers the opportunities for so called day traders to maximize their trading capital by avoiding the need to post overnight margins. When the trading becomes so easy, there is more and more on-line day trading over the Internet. The trading workflow is shown in Figure 10.13.

EXPERIMENT PURPOSE

We have tested level 3 application and have obtained a lot of cases. In this experiment, we test the adaptation capability of exception handling system.

EXCEPTION GENERATION

To generate exceptions, We removed a task from the system. When that removed task is asked to provide services, a null pointer exception will be generated because the task is not in the system.

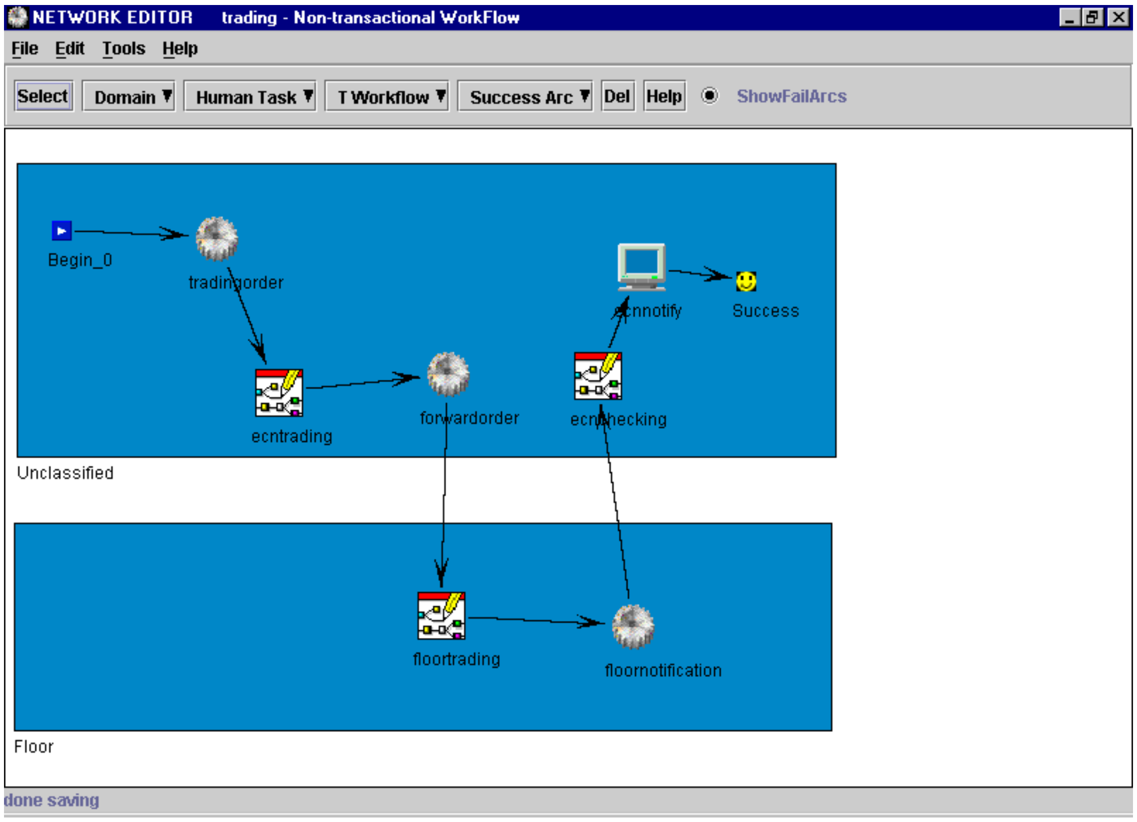


Figure 10.13 Future and stock trading application workflow

RESULT EVALUATION

We found that the adaptation capability of the cases is very limited. We have tested another workflow application with only one task in it. We stored the case in the case repository. When we tested this workflow application, we found that there was no adaptation at all. The solution provided by the old case will not work in the new situation. The following are the reasons we found:

- ORBWork is limited in adaptation. It is not easy to change the workflow. Human always needs to participate in the workflow changes. We don't want a solution that always needs a human to be present. High degree of automation is our goal.
- ORBWork currently does not provide query facility to query the specification and instance information about workflows, such as the preceding tasks and succeeding tasks for a given task, task activation parameters, etc. Because lack of this kind of information, it is hard for the exception handling system to generate an adaptable case.

In our original design, we did not consider these factors. We encountered failures of the exception-handling scheme derived by the exception handling system with almost every application we tried to test. After that, we designed a total new case structure in which there is an entry to denote whether the exception handling scheme is adaptable, and at what degree it is adaptable. In the CPR data structure, there is an entry called 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.

CONCLUSION AND FUTURE CONSIDERATIONS

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

SUMMARY

In this approach, the exception handling experience is shared among different organizations. For example, in the service subscription workflow, one of the exceptional situations is that credits history of the service requestor can not be verified, or it cannot be obtained in a timely manner. When this exception happens, the service subscription will often be delayed. If there is a good practice stored in the shared repository, by using the shared exception handling knowledge, the exception handling process can be very efficient.

Five exception handling coordination modes have been identified to meet the business needs. Business processes need coordination, so do the exception handling processes. For example, one of the five coordination modes is deferred exception handling. In this mode, when the service provider discovers that not enough money is supplied with the service request, the requested services will still be fulfilled. At a later moment, the service provider will raise an exception to the service requestor. This mode is proposed based on the principle of "customer satisfaction is the first priority".

The business processes are deployed to provide added values. They must be defined, maintained, and improved in alignment with the strategic management goals. Business process needs to be deployed flexibly as intended. To continuously improve the business process, the process must be well understood so its properties are evaluated, modifications are allowed, and future changes can be predicted. We view this continuous improvement process as a knowledge management assisted decision-making process. This process involves knowledge acquisition and problem solving. To achieve this goal, process histories are recorded and process knowledge is extracted. Process knowledge acquired will be used in incremental process improvement. Special attention is paid to process exceptions in this paper. While data mining over normal process working histories can extract business patterns and trend, exceptional situations usually directly denote when and where process improvement can be achieved.

Among many intelligent problem-solving techniques, we have identified that CBR is a promising technique. In implementing CBR as an intelligent problem solver for handling exceptions, we have found that it is necessary to combine default reasoning with CBR to achieve the results we need. In many situations, item values in the cases may be missing, or context information is not enough. Default reasoning helps solve this kind of problems.

A compensation preceding rework (CPR) is proposed as the exception-handling template in this dissertation in handling cross-organizational exceptions. The intelligent problem solving capabilities developed are used to populate this CPR template to adapt to the exception situations. That is, experience is shared, new situation is analyzed, and a new handler is derived. To automate the handling process, we have identified several modes of exception handling coordination. At the same time, we greatly value the importance of human involvement in the exception handling process. Thus, a GUI exception-handling client is developed so human beings can participate in the exception handling process.

We have conducted analytical assessment, use-cases assessment, and experimental assessment over our exception handling system. In analytical assessment, we have built a mathematical model to understand what a promising technique CBR is. By using use-cases assessment, We have identified the impact upon ORBWork WfMS by implementing our exception handling schemes. We have also identified several weakness points of the ORBWork workflow management system.

We have built five workflow applications to test our exception handling system. They are from three industry sectors: telecommunication, healthcare, and financial business. They are DSL service subscription application before and after the ACT of Telecommunication 1996, Level 3 IP network service workflow, infant transportation workflow, and stock and future trading workflow. We have generated more than 100 cases in testing our system.

The result of the assessments has shown us what a promising system our exception handling system is. It has also shown that the effectiveness of our exception handling systems is strongly co-related to the dynamic change capabilities of the 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 cases, 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

This work has three major future work directions. Since we only know the possible directions of our future work, we are going to list them here.

- Survivable computing and flexible process interactions. As identified in dissertation, current WfMSs can not support very flexible business processes. Future efforts need to be on this topic. However, it is always beneficial to identify to what extent the processes will be flexible. That is, what are the real needs from real world for a flexible mechanism to enact flexible processes? It is very rare in current literature to find descriptions on enactment for flexible business processes. That is why in our approach, we have identified five interconnection patterns for handling exceptions across organizational boundaries. Furthermore, since the case repository stores prior knowledge, it should be exploited how this knowledge can be used to enhance not only a system's exception handling capability, but also survivability and flexibility.
- Improving the system by learning from the exceptions. Once the cases have been obtained, we still need criteria for when and how the systems can be improved based on what we learned. That is, when should we modify the system based on these cases?
- 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.

REFERENCE

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

[Adam et al 1999] N Adam, O Dogramaci, A Gangopadhyay, Y. Yesha, Electronic Commerce, Technical, Business, and Legal Isues, Prentice Hall PTR, 1999

[Alonso et al 1999] G. Alonso, U. fiedler, C. hagen, A. Lazcano, H. Schuldt, and N. Weiller. Processes in lectronic Commerce. In ICDCS Workshop on Electronic Commerce and Web-based Applications, Austin, Texas, May 1999

[Baralist et al 1995] E. Baralist, S. Ceri, and S Paraboschi. “Improved Rule Analysis by Means of Triggering and Activation Graphs”, In Timos Sellis, editor, Proc. Of the Second Workshop on rules in Database systems, LNCS 985, P 165-181, Athens, Greece, September 1995

[Berry, Myers 1998] Pauline M. Berry, Karen L. Myers; "Adaptive Process Management: An AI Perspective", CSCW 1998, Towards Adaptive Workflow Workshop, Seattle, WA 1998

[Borgida 1999] A. Borgida, T. Murata, “Tolerating Exceptions in Workflows: a Unified Framework for Data and Processes”, Proceedings of the International Joint Conference on Work Activities coordination and Collaboration, WACC’99, February 22-25, 1999, San Francisco, CA

[Cichocki et al 1997] A. Cichocki, A. Helal, M. Rusinkiewicz, D. Woelk, “Workflow and Process Automation: Concepts and Technology”; Kluwer Academic Publishers ISBN 0-7923-8099-1 December 1997

[Cichocki et al. 97]A. Cichocki and M. Rusinkiewicz, Migrating Workflows, Advances in Workflow Management Systems and Interoperability, Istanbul, Turkey, August 1997.

[Ceri et al 1997] S. Ceri, P. Grefen, and G. Sanchez, "WIDE: A distributed architecture for workflow management", in Proceedings of RIDE 1997, Birmingham, UK, April 1997

[Carlsen and Jorgensen 1998] S. Carlsen and H. Jørgensen, Emergent Workflow: The AIS Workware Demonstrator, CSCW98, towards adaptive workflow systems, Seattle, WA 1998

[Casati et al 1998] Casati, F.; Ceri, S.; Pernici, B.; Pozzi, G.; Workflow Evolution. Data & Knowledge Engineering, 24(3), Jan. 1998, pp. 211-238

[Casati et al 1999] Fabio Casati, Mariagrazia Fugini and Isabelle Mirbel, An Environment for Designing Exceptions in Workflows, Information Systems, Vol. 24, No. 3, pp255-273 1999

[Ceri et al 1997] S. Ceri, P. Grefen, and G. Sanchez, "WIDE: A distributed architecture for workflow management", in Proceedings of RIDE 1997, Birmingham, UK, April 1997

[Cox and Reid 2000] D.R. Cox and N. Reid, The theory of the Design of Experiments, Chapman & Hall/CRC, 2000

[CrossFlow] CrossFlow. www.crossflow.org

[Das et al. +97] S. Das, K. Kochut, J. Miller, A. Sheth, and D. Worah, ORBWork: A Reliable Distributed CORBA-based Workflow Enactment System for METEOR$_2$, Technical Report UGA-CS-TR-97-001, LSDIS Lab, CS Department, Univ. of Georgia, February 1997

[Deiters et al 1998] Wolfgang Deiters, Thomas Goesmann, Katharina Just-Hahn, Thorsten Loffeler, Roland Rolles; Support for Exception Handling through

Workflow Management Systems, , CSCW, Towards Adaptive Workflow, Seattle, WA 1998

[ebXML 2000] ebXML, http://www.ebxml.org/, November 5, 2000

[Eder and Liebhart 1998] J. Eder and W. Liebhart, “Contributions to Exception Handling in Workflow Systems”, EDBT Workshop on workflow management Systems, Valencia, Spain, 1998

[Ellis et al. 95] C. Ellis, K. Keddara, and G. Rozenberg, Dynamic Changes within Workflow Systems in Proc. of the Conf. on Organizational Computing Systems (COOCS’95)}, 1995.

[Elmagarmid 1992] Elmagarmid, A., editor, Database Transaction Models for Advanced Applications, Morgan Kaufmann Publishers, Inc., San Mateo, CA 1992

[Forrester] www.Forrester.com

[Georgakopoulos et al 1995] D. Georgakopoulos, M. Hornick, and A. Sheth, “An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure”, Distributed and Parallel Databases, 3(2):119--154, April 1995

[Georgakopoulos et al 1999] D. Georgakopoulos, H. Shuster, A. Cichocki, and D. Baker. Managing Process and Service Fusion in Virtual Enterprises. Information systems, 1999

[Gray and Reuter 1993] Gray, J., Reuter, A.; Transaction Processing: Concepts and Techniques, Mogan Kaufmann Publisher, San Mateo 1992

[Guimaraes et al. 97] N. Guimaraes, P. Antunes, and A. Pereira, The Integration of Workflow Systems and Collaboration Tools, Advances in Workflow Management Systems and Interoperability}, Istanbul, Turkey, August 1997.

[Gulla et al 1991] Gulla, J. A., Lindland, O. I., and Willumsen, G., ãPPP - An Integrated CASE Environment,ä Third International Conference on Advanced Information Systems Engineering (CAiSE'91), Trondheim, Norway, 1991.

[Hagen and Alonso 1998] C. Hagen, G. Alonso, “Flexible Exception Handling in the OPERA Process Support System”, 18th International Conference on Distributed Computing Systems (ICDCS), Amsterdam, The Netherlands, May 1998.

[Han et al] Y. Han, A. Sheth, and C. Bussler, a taxonomy of Adaptive Workflow Management, CSCW98, towards adaptive workflow systems, Seattle, WA 1998

[Han 97] Y. Han, "HOON - A Formalism Supporting Adaptive Workflows," Technical Report #UGA-CS-TR-97-005, Department of Computer Science, University of Georgia, November 1997.

[Han and Sheth 98] Y. Han and A. Sheth, "On Adaptive Workflow Modeling," the 4th International Conference on Information Systems Analysis and Synthesis, Orlando, Florida, July, 1998

[Hermann 95] T. Hermann, Workflow Management Systems: Ensuring Organizational Flexibility by Possibilities of Adaptation and Negotiation, in Proc. of the Conf. on Organizational Computing Systems (COOCS’95)}, 1995

[IDSO 2000] CAiSE workshop on Infrastructures for Dynamic Business-to-Business Service outsourcing, Stockholm, June 5-6 2000

[Joeris and Herzog 1998] Joeris, G.; Herzog, O.: Managing Evolving Workflow Specifications. Proceedings of CoopIS'98, New York, August 1998

[Jablonski et al. 97] S. Jablonski, K. Stein, and M. Teschke, Experiences in Workflow Management for Scientific Computing, Proceedings of the Workshop on Workflow Management in Scientific and Engineering Applications (at DEXA97), Toulouse, France, September 1997.

[JFLOW] OMG jFlow Submission, ftp://ftp.omg.org/pub/docs/bom/98-06-07.pdf

[Kang et al 1999] Myong H. Kang, Judith N. Froscher, Amit P. Sheth, Krys Kochut, John A. Miller: A Multilevel Secure Workflow Management System. CAiSE 1999: 271-285

[Ketan 1999] Ketan Aroon Bhukhanwala, "JFLOW : workflow interoperability for the METEOR workflow management system", Thesis (M.S.), University of Georgia, 1999.

[Krishnakumar and Sheth 95] N. Krishnakumar and A. Sheth, “Managing Heterogeneous Multi-system Tasks to Support Enterprise-wide Operations,” Distributed and Parallel Databases Journal, 3 (2), April 1995

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

[Klein and Dellarocas 1998] 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

[Klein 2000] Mark 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

[Kochut et al 1999] 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

[Krishnakumer and Sheth 1995] 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

[Kulak and Guiney 2000] D. Kulak and E. Guiney, "Use cases: requirements in context", New York: ACM Press; Boston: Addison-Wesley, c2000

[Lazcano et al 2000] A. Lazcano, G. Alonso, H. Schuldt, C. Schuler: The WISE approach to Electronic Commerce. International Journal of Computer Systems Science & Engineering, special issue on Flexible Workflow Technology Driving the Networked Economy (to appear in September 2000).

[Leymann and Roller 2000] F. Leymann, D. Roller: Production Workflow: Concepts and Techniques, Upper Saddle River, Prentice Hall 2000

[Lin 97] C. Lin, “A Portable Graphic Workflow Designer,” M.S. Thesis, Department of Computer Science, University of Georgia, May 1997.

[Liu and Pu 1998] Liu, L., Pu, C.: Methodical Restructuring of Complex Workflow Activities. Proceedings of 14th Internatinal Conference on Data Engineering (ICDE' 98), Orlando, Florida, February 1998, pp. 342-350

[LSDIS 2000] workflow repository, LSDIS Lab report, University of Georgia, 2000

[Ludwig 1999] Ludwig, H, 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 122 - 129, IEEE Computer Society, Los Alamitos, 1999

[Ludwig and Whittingham 1999] 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, Februrary 22-25, 1999, San Francisco, CA

[Ludig et al. 1999] H. Ludig, C Bussler, M. Shan, P. Grefen, Cross-Organisational Workflow Management and Co-ordination Workshop Report, February 22nd 1999, San Francisco

[Luo et al 1998] Zongwei Luo, Amit Sheth, John Miller, Krys Kochut, “Defeasible Workflow, its Computation, and Exception Handling”, Proceedings of 1998 Computer-Supported Cooperative Work (CSCW 1998), Towards Adaptive Workflow Systems Workshop, Seattle, WA, 1998

[Luo 1999] Zongwei Luo, “Applying Workflow Technology in Trading Systems”, Proceedings of the 12th International Conference on Computer Applications in Industry and Engineering (CAINE-99), Atlanta, Georgia (November 1999)

[Luo 2000] Zongwei Luo, “Checkpointing for Workflow Recovery”, Proceedings of ACM Southeast Conference (ACM SE 2000), Clemson, SC, April 2000, pp79-80

[Luo et al 2000] Zongwei 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 13, Number 2, September/October, 2000, pp125-147

[Marek and Truszczynski 1993] V. Marek, M. Truszczynski, Non-monotonic Logic, Context-Dependent Reasoning, Springer-Verlag, 1993

[McClatchey et al. 97] R. McClatchey, J.-M. Le Geoff, N. Baker, W. Harris, and Z. Kovacs, A Distributed Workflow and Product Data Management Application for the Construction of Large Scale Scientific Apparatus, Advances in Workflow Management Systems and Interoperability}, Istanbul, Turkey, August 1997.

[Halvey and Melby 2000] John K. Halvey, Barbara Murphy Melby, Business process outsourcing : process, strategies, and contracts, John Wiley, New York,2000

[METEOR] METEOR project home page, http://lsdis.cs.uga.edu/proj/meteor/meteor.html

[METEOR Model 3] Krys Kochut, METEOR Model 3, Draft Proposal, Version 1.0, LSDIS Lab, Department of Computer Science, the University of Georgia, July, 1999

[Miller et al. 98] J. Miller, D. Palaniswami, A. Sheth, K. Kochut, H. Singh "WebWork: METEOR's Web-based Workflow Management System", Journal of Intelligent Information Systems, (JIIS) Vol. 10 (2), March 1998.

[Marek and Truszczynski 1993] V. Marek, M. Truszczynski, Non-monotonic Logic, Context-Dependent Reasoning, Springer-Verlag, 1993

[Reichert and Dadam 1998] M. Reichert, P. Dadam, ADEPTflex - Supporting Dynamic Changes of Workflows Without Loosing Control, Journal of Intelligent Information Systems (JIIS), Special Issue on Workflow Management Systems, Vol. 10, No. 2, pp. 93-129, March 1998

[Reichert and Dadam 98] M. Reichert and P. Dadam, ADEPT flex: Supporting Dynamic Changes of Workflows Without Losing Control, Journal of Intelligent Information Systems, 10 (2), March 1998.

[RosettaNet 2000] RosettaNet, www.rosettanet.org

[Saastamoinen1995] H. Saastamoinen, "On the Handling of Exceptions in Information Systems". PhD thesis, University of Jyvaskyla, 1995

[Sheth 97] Sheth, "From Contemporary Workflow Process Automation to Adaptive and Dynamic Work Activity Coordination and Collaboration," Proceedings of the Workshop on Workflows in Scientific and Engineering Applications, Toulouse, France, September 1997.

[Sheth et al. 96] A. Sheth, D. Georgakopoulos, S. Joosten, M. Rusinkiewicz, W. Scacchi, J. Wileden, and A. Wolf, Eds. Report from the NSF Workshop on Workflow and Process Automation in Information Systems, Technical Report UGA-CS-TR-96-003, Dept. of Computer Sc., University of Georgia, October 1996. http://lsdis.cs.uga.edu/lib/lib.html

[Sheth et al 97] A Sheth,D. Worah, K. Kochut, J. Miller, K. Zheng, D. Palaniswami, S. Das, "The METEOR Workflow Management System and its use in

Prototyping Healthcare Applications", Proceedings of the Towards An Electronic Patient Record(TEPR'97) Conference, April 1997, Nashville, TN.

[Sheth et al 1996] A. Sheth, K. J. Kochut, J. Miller, D. Worah, S. Das, C. Lin, D. Palaniswami, J. Lynch, and Shevchenko. “Supporting State-Wide Immunization Tracking using Multi-Paradigm Workflow Technology”, In Proc. of the 22nd. Intnl. Conference on Very Large Data Bases, Bombay, India, September 1996

[Sheth and Kochut 98] A. Sheth and K. Kochut, “Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems,” in A. Dogac, L. Kalinechenko, T. Ozsu and A. Sheth, Eds. Workflow Management Systems and Interoperability, NATO ASI Series F, Vol. 164, Springer Verlag, 1998.

[Sheth et al 1999] A.P. Sheth, W.M.P. van der Aalst, and I.B. Arpinar. "Processes Driving the Networked Economy: Process Portals, ProcessVortex, and Dynamically Trading Processes". IEEE Concurrency, 7(3): 18-31, 1999.

[Strong and Miller 1995] D. Strong and S. Miller, “Exceptions and exception handling in computerized information processes”, ACM Trans. Information System, 13, 2 (Apr. 1995)

[SWAP] Simple Workflow Access Protocol home page, http://www.ics.uci.edu/~ietfswap/index.html

[Taylor 97] R. Taylor, Endeavors: Useful Workflow Meets Powerful Process, Information and Computer Science Research Symposium, University of California at Irvine, February 1997. URL: http://www.ics.uci.edu/endeavors/

[tpaXML 2000] IBM tpaXML, http://www.ibm.com/developer/xml/tpaml/b2b-integration-with-tpa.pdf, November 6, 2000

[Van Der Aslst 1999] W. van der Aslst. Inter-organizational workflows: an Approach Based on Message Sequence Charts and Petri Nets. Information Systems, 1999

[Voorhoeve and Aalst 1997] M. Voorhoeve, W. Aalst, “Ad-hoc Workflow: Problems and Solutions”, 36-40, DEXA Workshop 1997

[Wedekind 1997] H. Wedekind, “Specifying Indefinite Workflow Functions in Ad-hoc Dialogs”, 30-35, DEXA Workshop 1997

[Weikum and Schek 1992] Concepts and Applications of Multi-level Transactions and Open-Nested Transactions, in [Elmagarimid 1992] chapter 13.

[WfMC] Workflow Management Coalition Standards, http://www.aiim.org/wfmc/mainframe.htm

[Willumsen et al 1993] Willumsen, G., Gulla, J. A., Lindland, O. I., and Seltveit, A. H., An Integrated Environment for Validating Conceptual Models, The 6th International Workshop on Computer-Aided Software Engineering (CASE'93), Singapore, 1993

[Worah et al 97] D. Worah, A. Sheth, K. Kochut, J. Miller, "An Error Handling Framework for the ORBWork Workflow Enactment Service of METEOR," Technical Report, LSDIS Lab. Dept. of Computer Science, Univ. of Georgia, June 1997.

[Worah and Sheth 1997] D. Worah and A. Sheth, “Transactions in Transactional Workflows”, in Advanced Transaction Models and Architectures, edited by S. Jajodia and L. Kerschberg, Kluwer Academic Publishers, Boston, August 1997

[Yong 98] J. Yong, "The Respository system of METEOR workflow management system", Master Thesis, Department of Computer Science, University of Georgia, March 1998.

[Zhang et al. 1994] Zhang, A., Nodine, M., Bhargava, B., and Bukhres, O., Ensuring Relaxed Atomicity for Flexible Transactions in Multi-database Systems, In

Proceedings of 1994 SIGMOD International Conference on Management of Data, pages 67-78

[Zheng 97] K. Zheng, Designing Workflow Processes in METEOR$_2$ Workflow Management System M.S. Thesis, LSDIS Lab, Computer Science Department, University of Georgia, June 1997.

[Zhou and Venkatesh 1999] Mengchu Zhou, Kurapati Venkatesh, "Modeling, simulation, and control of flexible manufacturing systems: a Petri net approach", Singapore; [River Edge], NJ: World Scientific, 1999

APPENDIX

PROCESS ANALYSIS

In our approach, a process system is considered as a reactive system that maintains an ongoing interaction with its environment. It is assumed the all variables that describe the properties of processes are taken from a set of variables, called process variable set or vocabulary. Instances of those variables form the process environment. Situation of the activities in processes at a certain point of time is called a process state that is specified through task states and data states and the status of workflow environment. Inter-state dependence constraints are enforced through *process transitions* that are specified in JECA rules [Luo et al 2000]. Thus, each process state S is associated with a JECA rule set R specifying process transitions. An initial process state is where a process starts. It has a special incoming transition, specified by a special JECA rule, called initial rule. The initial rule only triggers those rules associated with initial process states. A final process state is where a process terminates. It is associated with a special JECA rule, called final rule. The final rule cannot trigger any rules. A *process* is a series of process states linked by process transitions, starting from an initial process state, ending at a final process state.

Consider a JECA rule r and the process transition δ specified by r. Given JECA rules r_i, r_k and $r_{j,}$ if r_i triggers r_k and r_k triggers $r_{j,}$ then r_i transitively triggers $r_{j.}$ Given JECA rules r_1, r_2, …, r_n, (n>=3) r_1 transitively triggers r_n if r_i triggers r_{i+1}, 1<= i <n. A process execution sequence consists of a series of process states linked by triggered JECA rules. The process execution is said to be in a *deadlock* if the last process state is a not final state. A process execution sequence is an execution history of a process

instance. Each process instance is associated with a process specification (also called process type).

TERMINATION

In rule execution, there is a danger of non-termination. We give a sufficient condition for the termination of rule execution over a JECA rule set [Luo et al 2000]. We ensure termination of evaluation of J and C in JECA rule evaluation, we limit the logic expressions used in specifying J and C components of JECA rules to be quantifier free. Furthermore, we assume that facts that need to be checked in rule evaluations are finite. Petri net has been extensively used as a verification tool. Various system properties such as termination, liveness, boundness, et al. are verified using this standard tool [Zhou and Venkatesh 1999]. To avoid any confusion about this JECA rule modeling with common predicate rule modeling, we utilize an approach of activity graph [Baralist et al 1995] to analyze business processes similar to Petri net. As known, in Petri net, predicate rule evaluation is usually not considered. Similarly in our research, we put this issue aside, that is, how to evaluate predicate rule will not be our concern here.

Consider a JECA rule r (j, e, c, a). When event (e) occurs, r is triggered. In other words, event (e) triggers JECA rule r. Action of JECA rules can generate events that can trigger other JECA rules, which may include themselves. The action of those triggered JECA rules may further trigger more JECA rules. This series of JECA rules triggering forms a triggering graph. Consider an arbitrary JECA rule set R. Triggering graph (TG) over R is a directed graph where each node corresponds to a rule r_i that belongs to R, and a directed arc (r_i, r_k) means that the execution of rule r_i generates events that trigger rule r_k.

Consider a JECA rule set R, and TG over R. An irreducible rule set over R is a subset of R, and includes only those JECA rules whose incoming arcs are in the directed graph, TG. This irreducible rule set is generated by iterations of discarding a rule that does not have an incoming arc in TG, and remove all its outgoing arcs. The iterations continue until all the rules have been removed, or until all the remaining rules have incoming arcs in the entire directed graph, TG. If the irreducible rule set over R is empty, then rule execution on R is guaranteed to terminate. This is a sufficient condition for termination of rule execution over rule set R. Assume that rule execution will not terminate if the irreducible rule set over R is empty. If rule execution will not terminate, there are always triggered rules. There must exist at least one triggering cycle involving the same rules, say r_1 and r_2, in the same direction. That is, r_1 trigger r_2. Thus, both r_1 and r_2 have incoming arcs in the directed graph, TG. So the irreducible set over R is not empty. This contradicts with the assumption.

REACH-ABILITY

An important issue in process execution is whether a process can reach a specific process state. In order to find out whether a modeled business process can reach a specific process state as a result of a required functional behavior, we developed the process graph. It is used to find such a sequence of triggering of transactions that would transform a process state p_0 to p_i, where pi represents this specific state.

A process state p_i is said reachable from a process state m0 if there exist a sequence of transitions that transforms a process state p_0 to p_i. A process state p_1 is said to be immediately reachable from p_0 if firing a triggered transition in p_0 results in p_1.

Consider a process specification, and associated JECA rule set R. Process graph (PG) is a directed graph, where

- each node corresponds to a rule r_i belongs to R.
- A directed arc (r_i, r_k) means rule r_i enables rule $r_{k.}$
- If r_i in a direct arc (r_i, r_k) does not have incoming arcs, r_i is the initial rule.
- If r_k in a direct arc (r_i, r_k) does not have outgoing arcs, r_k is the final rule.
- Initial rule transitively enables final rule.
- Irreducible rule set obtained from R is empty.

A process specification can only be correct if all possible process execution sequences start from an initial process state, and end at a final state. Consider a process specification, and associated JECA rule set R. If a process graph exists, the initial rule transitively triggers the final rule. All process execution sequences can only start from initial process state because the initial rule is the only node in the process graph that does not have any incoming arcs, but has outgoing arcs. Similarly, all process execution sequences can only end at final process state because the final rule is the only node in the process graph that does not have any outgoing arcs, but has incoming arcs. Since irreducible rule set obtained from R is empty, rule execution is guaranteed to terminate. Thus, all process execution sequences can only start from an initial process state, and will end at a final state. So if a process graph can be obtained from its associated process specification, the specification is correct.

BOUNDED-NESS

In process support systems, work-list server holds the working items. The capacity of these servers to hold these working items are usually constrained, e.g. memory constraint. It is important to be able to determine whether proposed process execution strategies can prevent from the violation of the constraints, e.g., memory overflow. To identify the existence of overflow in the modeled business process, the bounded-ness of the process must be checked. To support bounded-ness check, a token is used to associate with each process activity and a weight is used to associate with each process transition. The number of working items for a process activity p_a is said to be k-bounded if the number of tokens associate with p_a is always less or equal to k for every process state p reachable from the initial state p_0.

LIVE-NESS

A business process must be live. The concept of live-ness is closely related to the deadlock situation. Four conditions that must hold for a deadlock to occur are mutual exclusion of resources, resource hold and wait, no preemption, and circular wait. A general requirement for a deadlock free process is that for all process state p, which are reachable from the initial process state p_0, it is ultimately possible to fire any transition in the process by progressing through some firing sequence. The live-ness criterion we employ in analyzing business processes is:

- A process specification can only be live if all possible process execution sequences start from an initial process state, and end at a final state.

REVERSIBILITY

An important issue in the process supporting systems is the ability of these systems for an error recovery. In case of errors or failure, these systems are required to recover from the failure states and return to the preceding correct states or equivalent state. A business process, with the initial process state p_0, is said to be reversible if for each process state p reachable from p_0, p_0 is reachable from p.

This reversibility requirement is sometimes too restrictive. A less restrictive and more practical requirement is to find a state p_i that for each marking p reachable from p_0, p_i is reachable from p.

SYSTEM DEMONSTRATION

Here we are using the level 3 telecommunication application to show an exception-handling scenario. In this application, 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. More information about this application can be found in Chapter 1.

In this demonstration, 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. The main purpose of the demonstration is to show how a case adapts to a new situation, i.e., how a case is reused.

MECHANISM FOR CASE REUSE

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 called NULL adaptation. More information about case reuse can be found in Chapter 5 and [Luo et al 2000].

DEMONSTRATION STEPS

In this demonstration, we first build the level 3 application, compile it, and then deploy it. Inside the level 3 application, 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 demonstration:

1. Deploy the level 3 application.
2. Create a new instance for this application.
3. 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.
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. Now 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: I3

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. You will see 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.