Applied Intelligence 13, 125–147, 2000

Exception Handling in Workflow Systems

ZONGWEI LUO, AMIT SHETH, KRYS KOCHUT AND JOHN MILLER
Large Scale Distributed Information System Lab, 415 GSRC, Computer Science Department, University of Georgia Athens, GA 30602, USA*
luo@ainge.cs.uga.edu
amit@ainge.cs.uga.edu
kochut@ainge.cs.uga.edu
jam@ainge.cs.uga.edu

Abstract. In this paper, defeasible workflow is proposed as a framework to support exception handling for workflow management. By using the "justified" ECA rules to capture more contexts in workflow modeling, defeasible workflow uses context dependent reasoning to enhance the exception handling capability of workflow management systems. In particular, this limits possible alternative exception handler candidates in dealing with exceptional situations. Furthermore, a case-based reasoning (CBR) mechanism with integrated human involvement is used to improve the exception handling capabilities. This involves 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.

Keywords: case-based reasoning (CBR), context-dependent reasoning, exception handling, ontology, workflow system

1. Introduction

Workflow technology is considered nowadays as an essential technique to integrate distributed and often heterogeneous applications and information systems as well as to improve the effectiveness and productivity of business processes [1–4]. A workflow management system (WfMS) is a set of tools providing support for the necessary services of workflow creation, workflow enactment, and administration and monitoring of workflow processes, which consist of a network of tasks. These workflow processes are constructed to conform to their workflow specifications. However, due to foreseen or unforeseen situations, such as system malfunctions due to failure of physical components or changes in business environment, deviations (exceptions) of those workflow processes from their specifications are unavoidable. Workflow systems need exception handling mechanism to deal with those deviations. Exception-handling constructs, as well as the underlying mechanisms, should be sufficiently general to cover various aspects of exception handling in a uniform way. In addition, it helps to separate the modules for handling exceptional situations from the modules for the normal cases. We believe that designing an integrated human-computer process may provide better performance than moving toward an entirely automated process in exception handling.

Let's take a look at a motivating workflow application to characterize the scope of exception handling (see Fig. 1). 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

*http://lsdis.cs.uga.edu

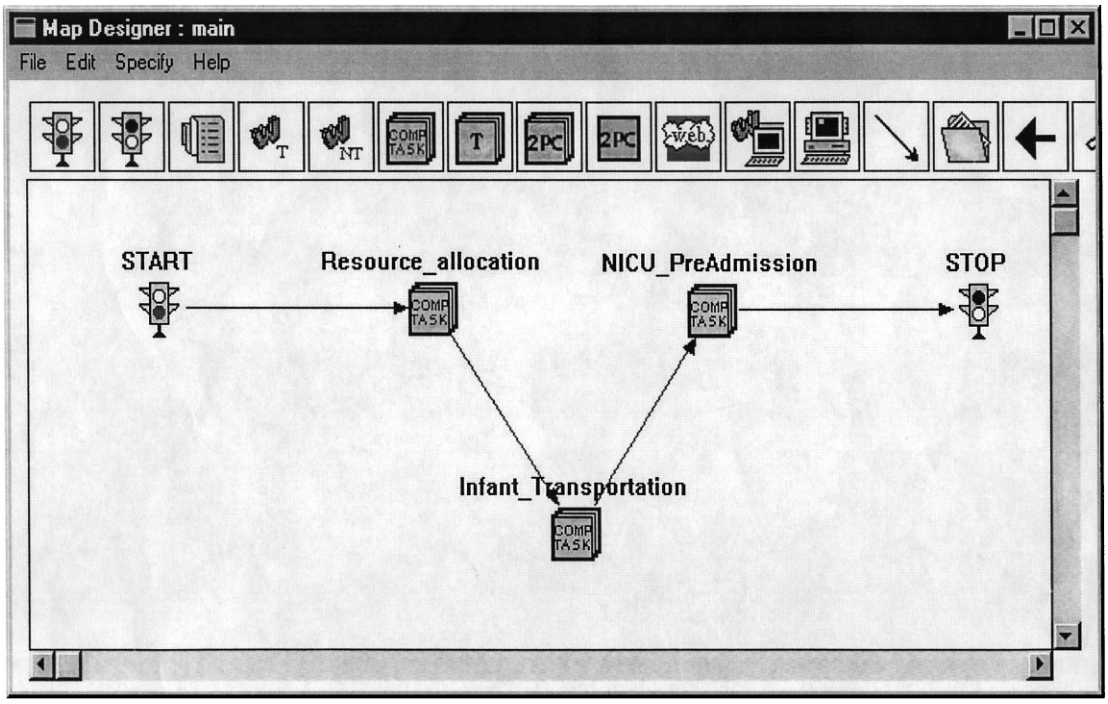

Figure 1. High-level workflow for newborn infant transportation.

there are two or three healthcare professionals who perform different roles. During the transport the ambulance personnel perform the standard procedures to obtain medical data for the infant. The first step of this application involves the task that allocates resources, such as healthcare professionals, equipment, and so on. The last step is to prepare for admission to the NICU.

If exceptions are raised during the transport, corrective actions must take place. The decision of the corrective procedures 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 at the top of Fig. 2. 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 (as shown at the bottom of Fig. 2).

This healthcare application raises several requirements for workflow systems to support:

- The processes in this application are very dynamic. To support coordination of such processes, a systematic way of workflow evolution, including dynamic structure modifications should be worked out.
- There are potential collaboration activities in this transport process; e.g., the healthcare professionals may need advice from specialists in the NICU. The advice is context based. The results from the collaborations may affect the progress of the ongoing process coordination.
- 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) [5] based exception-handling system stores valuable experience learned to help them make decisions.
- 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

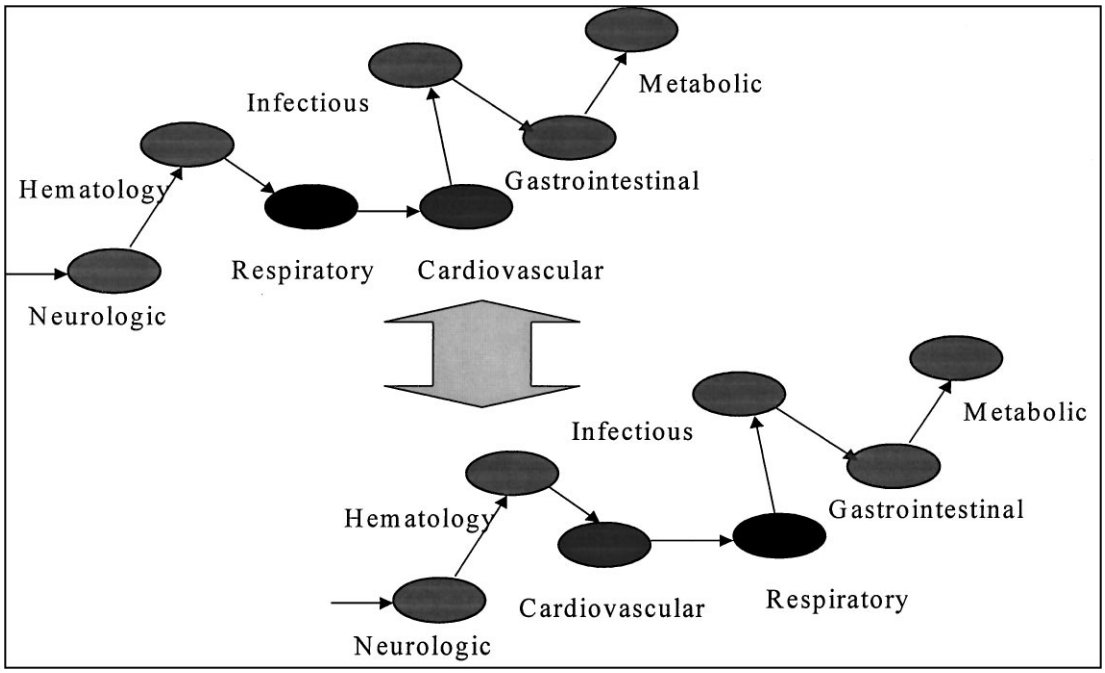

Figure 2. Treatment plan change in infant transportation task.

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 workflow execution context.

Approaches to address the first two requirements, although topics of our research are beyond the scope of this paper. In this paper, we discuss an approach of defeasible workflow to address the last two requirements. Defeasible workflows target workflow applications in the dynamic and uncertain business environment by modeling organizational processes through justified ECA (JECA) rules developed to support context-dependent reasoning processes. There are three major contributions to exception handling in workflow management systems in this approach.

1. By using the JECA rules to capture more contexts in workflow modeling, defeasible workflow enhances the exception handling capabilities of WfMSs through supporting context dependent reasoning in dealing with uncertainties. It can limit possible alternative exception handlers in dealing with exceptional situations.
2. It considers workflow evolution, which is an active research topic, a good candidate to exception handling, which is usually called adaptive exception handling. That is, possible modifications of workflows are considered as exception handlers, along with other candidates such as ignore, retry, workflow recovery, and so on. Thus, several modification primitives are given in defeasible workflow that will ensure the modified workflows meet the correctness criteria established.
3. A case-based reasoning (CBR) [5] based exception handling mechanism with integrated human involvements is used in defeasible workflow 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.

The organization of this paper is as follows: We give a perspective on exceptions in Section 2 that provides directions on how exception aware systems should be built. We introduce defeasible workflow in Section 3 that is used to support such healthcare applications

like the infant transport. In Section 4, we introduce a CBR-based approach for the exception handling. In Section 5, we discuss related works. Finally, Section 6 concludes this paper.

2. What is an Exception?

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 interests 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 Fig. 3, 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 those points.

- *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, the decision 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 be adapted 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

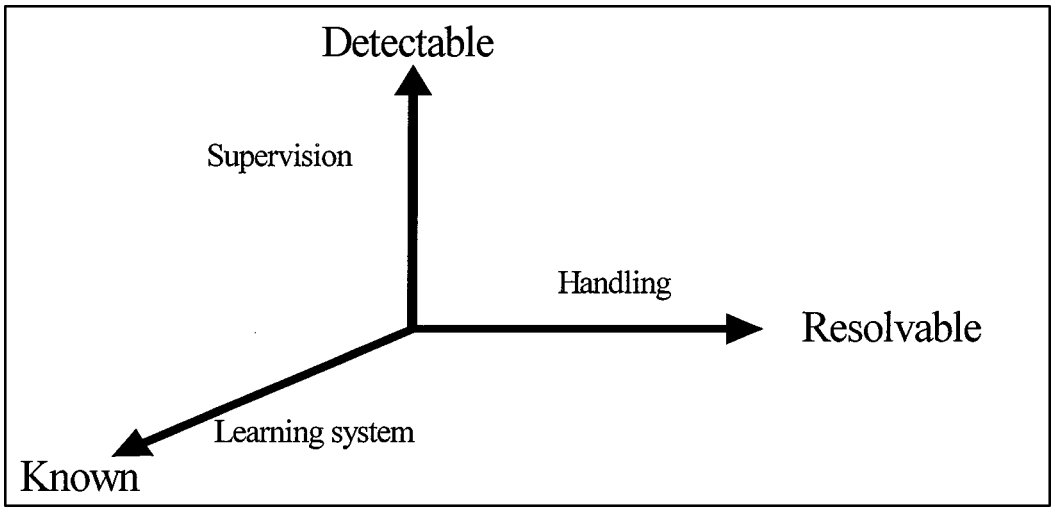

Figure 3. Three-dimensional analyses of exceptions.

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 capable exception handlers 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 for 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, 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 exceptions occur, but the system cannot derive a solution to solve the deviation to meet the requirement, then it is called irresolvable exceptions.

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. Defeasible workflow is proposed in Section 3 to support a systematic way of propagation and masking.

3. Defeasible Workflow

Organizational processes are often dynamic. They evolve over time and often involve uncertainty. To adapt to its environment, a workflow should be flexible enough that necessary modifications to its 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. For example, uncertainties in clinical decision-making processes, such as incomplete data in the report, should be eliminated as early as possible, preferably at the design stage. However, if this is not possible at runtime exceptions should be raised and handled. The clinical decision-making is a process by which alternative strategies of care are considered and selected. Those alternatives can be potentially limited if more useful contexts are available during the clinical decision-making. Furthermore, the following basic facts about clinical decision processes [6] have shown that in very complicated situations, necessary contexts should be captured in order to make correct decisions efficiently:

- "Human and environmental influences are frequently complex, uncertain and difficult to control." Necessary context can help analyze the complex situations.
- "Decisions should be clear-cut and error-free. The nursing and medical professions have expressed an interest in the precision and objectivity by which decisions are made, while simultaneously retaining an individualistic, holistic approach to patient management." The context-dependent reasoning approach is one of the solutions to support such decision-making processes.
- "Standardization of management plans is usually accompanied by paying less attention to differences in patients' needs." This actually means such standardization should consider individual differences that can only be achieved by adding exceptions to the standardization. Those exceptions are actually used

to capture necessary context when the standardized plans are enforced.

Defeasible workflows target workflow applications in the dynamic and uncertain business environment by capturing more workflow contexts and supporting context-dependent reasoning processes. They are characterized as follows:

- Organizational processes are modeled through JECA rules (see below).
- When no default consequences can be derived during the execution, or the conflicts occur that cannot be resolved in the default evaluation, exceptions will be raised. WfMSs will try to derive capable handlers to handle the raised exceptions.
- If none of the derived handlers are suitable, or handlers cannot be derived, past experiences in handling exceptions will be sought and reused by retrieving and adapting similar exception handling cases stored in the case repository.
- Human interaction will be necessary if no acceptable solutions can be automatically derived. Solutions provided by a person, who is usually a specialist, will be abstracted, and stored into the case repository.

3.1. JECA Rules

We have extended the well known ECA rules specification as Justified Event-Condition-Action (JECA) rules to model business logic based on work in context-dependent reasoning [7]. There are several workflow prototypes (e.g. [8, 9]) that have adopted ECA rules as modeling tools. However, the contexts that can be captured by ECA rules are limited. The C in an ECA rule, which is used to capture rules evaluation context, is used as a condition that should be satisfied so that the action specified in that ECA rule can be executed. That is, an ECA rule, once triggered, can only be denied if its condition cannot be satisfied. This makes ECA rules incapable in modeling workflows in uncertain business environments. In JECA rules, justification (J) provides a reasoning context for the evaluations of ECA rules to support context dependent reasoning processes in dealing with uncertainties. Each JECA rule *r* (j, e, c, a) consists of four parts as follows:

- Justification (j): Justification forms the reasoning context in which evaluation of the specific JECA rule to be performed. Usually it is specified as a disqualifier, i.e., a JECA rule will be disqualified if its justification is evaluated true.
- Event (e): when event occurs, related JECA rules will be evaluated. We say the JECA rules are triggered.
- Condition (c): logic constraints to be satisfied so the action in the rule can be taken if the rule is not disqualified.
- Action (a): necessary actions to be taken if this rule is not disqualified, and events occur, conditions are met.

Consider a simple rule-processing algorithm given in Algorithm 3.1. This algorithm executes JECA rules by picking up a triggered rule one by one. When there are no more triggered rules, execution will terminate.

while there are triggered JECA rules do:

1. find a triggered JECA rule r
2. evaluate r's condition and justification
3. if r's condition is true, and justification is not true then execute r's action

Algorithm 3.1 A simple JECA rule execution algorithm

Consider the following example of exception handling rule in dealing with an exception related to cardiac diseased. In this rule, the exception is related to certain type of cardiac disease d, and the corrective action is to take an initial assessment for this type of cardiac disease d. The justification provides a reasoning context (as commonly followed in this medical specialty) that if there are one or more blood family members of opposite sex who had this type of cardiac disease d before, then this disease cannot be this type of cardiac disease d. The initial assessment can only be performed if the justification doesn't disqualify this JECA rule.

Event: Cardiac disease d related exception event
Condition: Risk factor is heart murmurs (related to the type Cardiac Disease d)
Action: Initial Assessment for Cardiac Disease d
Justification: Blood family member of opposite sex does have this type of Cardiac disease d

The above example, if modeled in ECA rules, can be:

Event: Cardiac disease d related exception event
Condition: Risk factor is heart murmur (related to the type Cardiac Disease d) and Blood family member of opposite sex doesn't have this type of Cardiac disease d
Action: Initial Assessment for Cardiac Disease d.

When a cardiac disease d related exception event actually occurs, necessary contexts should be available so that the condition in that ECA rule can be checked, i.e., the risk factor is heart murmur and blood family member of opposite sex does have this type of cardiac disease. If the condition "*Blood family member of opposite sex does have this type of cardiac disease d*" cannot be checked true, then this ECA rule cannot be satisfied. Is it correct not to model that condition of "*Blood family member of opposite sex does have this type of cardiac disease d*" in the ECA rule? The ECA rule then becomes:

Event: Cardiac disease d related exception event
Condition: Risk factor is heart murmur (related to the type Cardiac Disease d)
Action: Initial Assessment for Cardiac Disease d.

However, the answer is no, because this ECA rule cannot model that the action should not be executed when condition "*Blood family member of opposite sex does have this type of cardiac disease d*" is true. In the approach of JECA rules, the condition of "*Blood family member of opposite sex does have this type of cardiac disease d*" is a justification for that JECA rule. If the justification cannot be evaluated true, then the JECA rule will not be disqualified. So the action in that JECA rule can be executed if the risk factor of heart murmur is true.

3.2. JECA Rules Evaluation

To apply the context-dependent reasoning processes [7], those JECA rules are transformed into a form that is usually used in the context-dependent reasoning, called *default* [7]. A default has three parts, *prerequisite* (P), *justification* (J), and *consequence* (C) specified in logic expressions. Those defaults are constructed through JECA rules with mappings from J to justifications, C to prerequisite and A to consequence. Usually E in a JECA rule is associated with the justifications and prerequisite in the default that the JECA rule is mapped to. In order to apply the defaults, the prerequisite of the defaults should be proved. It is required the defaults should be *justified*. This means the justifications should not deny the applicability of the default. Consequence, the third part in default, forms a belief set through extension computation [7]. Please refer to [7] for detailed information about extension computation. Conflicts such as inconsistencies existing in the belief set should be resolved by non-monotonic reasoning process in which reasoning results may become invalid if more facts become available, or reasoning context has changed. An algorithm of execution of JECA rules through default evaluation is given in Algorithm 3.2.

while there are triggered defaults do:

1. find related defaults
2. evaluate defaults through context dependent reasoning, resolve any conflicts in the belief set
3. actions in the belief set will be executed

Algorithm 3.2 JECA rule processing through default evaluation

The defaults will be clustered into several sets called domains. Each default is associated with a local reasoning unit. If the local reasoning unit cannot make an acceptable decision, another reasoning unit at that domain that can get access to domain context will be consulted. A global reasoning unit is responsible for decision making in the global context. The local reasoning unit can also make a decision according to the partially available information without consulting another local unit or a unit at a higher level. This clustered reasoning architecture is well suited in workflow management because organizational processes modeled by workflows are clustered and/or layered in nature in which local decisions can often be reached. The criteria used in clustering can be either one or a combination of the following:

- Security: For security reasons, defaults must be clustered so sensitive information can be exchanged in a controlled manner. In the infant transport example certain medical records that may contain sensitive information, such as HIV, abnormal behavior of the infant's parents, should be securely controlled.
- Organization: It is natural to cluster defaults into several domains to model the organizational structure in that organization. In the infant transport example the

defaults can be grouped according to the organizational roles those health professionals can play.
- Performance: To provide better performance, it is necessary to cluster the defaults into several domains in which defaults can be more efficiently found and evaluated.

3.3. Conflict Resolution

During JECA rules evaluation, dynamic resolution is used to select default values (resources, algorithms, routing, numerical values, etc.) and resolve conflicts. Inconsistent information as well as coarse rule granularity in JECA rules can lead to conflicts during default evaluation. The conflicts may lead to exceptions if the conflicts cannot be resolved in the default evaluation. Conflicts in default evaluation can be either one or more of the following:

- Temporal conflict: defaults formed in subsequent time result in conflicts. For example, the healthcare professionals make two decisions separately. The decisions are specified in defaults that are fed into the workflow systems in the order the decisions are made. It is possible that the two decisions may result in conflicts. The first decision may involve certain kinds of drugs to be applied. The second decision may involve drugs that should not be applied after the drugs in the first decision are applied.
- Role conflict: defaults formed or evaluated by different role may result in conflicts. The healthcare professionals on board may have different opinions about the situation of the infant. During the assignment of healthcare professionals to the transport of infant, there may exist role conflicts, such as agents not available, an agent who can play that the caregiver role is not suitable to attend the infant.
- Semantic conflict: An example that might lead to semantic conflicts is alternative interpretation of the consequence of defaults. This is possible due to the nature of non-monotonic reasoning that is context dependent. If the contexts are different, interpretations in different contexts may also be different.

To resolve those conflicts, the defaults are assigned priorities, or more generally, are ordered by partial ordering relations regulated in business policies (organizational, security, etc.). That is, some defaults may be preferable to others, and assuming they are applicable, they should be used first. To resolve any inconsistencies, polices are based on one of the following criteria:

- Special: Preference will be given to exceptions over normal cases. For example, whenever a local decision cannot be made in the default evaluation, an exception is always raised.
- Hierarchical: Consequences derived at higher positions in the hierarchy will be favored over those at lower positions. In the infant transport application, if the medical situation of the infant becomes serious, the decision made from the personnel at a higher position who can be responsible for the action taken will be preferred.
- Temporal: Consequences from latest defaults will be favored over earlier ones. In the infant transport application, the personnel on board may consult the experts in the MCG's care unit (NICU). During the decision making process, the experts in the NICU might give one suggestion at one time, and later they might come up with another suggestion. The later one might be given a higher priority during the decision making process.
- Semantic: Interpretations that are more plausible will be preferred to less plausible ones. In the example application, if a symptom is discovered and if it might be caused by several different reasons, the personnel on board may derive one that he believes is the most reasonable.

3.4. Rule Graphs

In rule execution, there is a danger of non-termination. In this section, we give a sufficient condition for the termination of rule execution over a JECA rule set by extending the rule graphs in [10]. Furthermore we adapt the results from rule analysis [11] for workflow modeling. Further details about rule analysis related proofs appear in [10, 11]. 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.

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 two JECA rules r_i (j, e, c, a) and r_j (j, e, c, a). When condition (c) of r_i is true, JECA rule r_i is activated. If action (a) of r_j can change the condition (c) of r_i from false to true, we say r_i is activated by r_j, or r_j activates r_i. When justification (j) of r_i is not true, JECA rule r_i is justified. If action (a) of r_j can not change the justification (j) of r_i from false to true, we say r_i is justified by r_j, or r_j justifies r_i. Consider an arbitrary JECA rule set R. Activation graph (AG) 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 rule r_i activates rule r_k. Justification graph (JG) 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 rule r_j justifies rule r_k.

Consider a JECA rule set R, and TG, AG and JG over R. An irreducible rule set over R is a subset of R, and includes only those JECA rules whose incoming arcs are in all the directed graphs, TG, AG and JG. This irreducible rule set is generated by iterations of discarding a rule that does not have an incoming arc in TG, or AG, or JG, and remove all its outgoing arcs. The iterations continue until all the rules have been removed, or until al the remaining rules have incoming arcs in all the directed graphs, TG, AG, and JG. If the irreducible rule set over R is empty, then rule execution on R is guaranteed to terminate [11]. 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, according to the rule execution algorithm in Algorithm 3.1 or 3.2, there are always triggered rules, and some of those triggered rules' condition must be true and justification must not be true. There must exist at least one triggering cycle, activation cycle and justification cycle involving the same rules, say $r1$ and $r2$, in the same direction. That is, $r1$ trigger $r2$, $r1$ activates $r2$, and $r1$ justifies $r2$. Thus, both $r1$ and $r2$ have incoming arcs in all directed graph, TG, AG, and JG. So the irreducible set over R is not empty. This contradicts with the assumption.

3.5. Workflow Modeling

In our approach, a workflow 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 workflows are taken from a set of variables, called workflow variable set or vocabulary. Instances of those variables form the *workflow environment*. Situation of the activities in workflows at a certain point of time is called a *workflow state* that is specified through task states and data states and the status of workflow environment. Inter-state dependence constraints are enforced through *workflow transitions* that are specified in JECA rules. Thus, each workflow state S is associated with a JECA rule set R specifying workflow transitions. An initial workflow state is where a workflow 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 workflow states. A final workflow state is where a workflow terminates. It is associated with a special JECA rule, called *final rule*. The final rule cannot trigger any rules. A *workflow* is a series of workflow states linked by workflow transitions, starting from an initial workflow state, ending at a final workflow state.

Consider a JECA rule r and the workflow transition δ specified by r. If r is triggered, activated, and justified, r is enabled. δ is enabled if r is enabled. Given JECA rules r_i, r_k and r_j, if r_i enables r_k and r_k enables r_j, then r_i transitively enables r_j. Given JECA rules $r_1, r_2, \ldots, r_n$, $(n >= 3)$ r_1 transitively enables r_n if r_i enables r_{i+1}, $1 <= i < n$. A workflow execution sequence consists of a series of workflow states linked by enabled JECA rules. The workflow execution is said to be in a *deadlock* if the last workflow state is not final. A workflow execution sequence is an execution history of a *workflow instance*. Each workflow instance is associated with a *workflow specification* (also called *workflow type*).

Consider a workflow specification, and associated JECA rule set R. Workflow graph (WG) 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 workflow specification is correct if all possible workflow execution sequences start from an initial workflow state, and end at a final state. Consider a workflow specification, and associated JECA rule set R. If a workflow graph exists, the initial rule transitively enables the final rule. All workflow execution

sequences can only start from initial workflow state because the initial rule is the only node in workflow graph that does not have any incoming arcs, but has outgoing arcs. Similarly, all workflow execution sequences can only end at final workflow state because the final rule is the only node in workflow 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 workflow execution sequences can only start from an initial workflow state, and will end at a final state. So if a workflow graph can be obtained from its associated workflow specification, the specification is correct.

3.6. Exception Modeling

To handle exceptions, JECA rules are used to model the exception handling process. Event part captures exceptional events. Condition and justification part identifies the context. Action part specifies the operations to handle the exceptions. Possible operations that can be specified in the action part are as follows.

- Masking. It includes corrective actions such as ignore, retry, workflow recovery operations, modifications of workflows, etc.
- Propagation. It sends out warnings and/or propagates the exceptions.
- Recording. It records the exception situations for future reference

We characterize the types of exceptions into three broad categories (see Fig. 4) based on our early work in error handling [12]. The infrastructure exceptions and application exceptions are mapped to workflow exceptions that include system exceptions and user exceptions. That is, an exception that occurs at infrastructure layer will trigger a mapped exception at workflow layer. The same is true for an application exception. This mapping scheme is adopted due to the heterogeneous nature of exceptions in applications and infrastructure.

- Infrastructure exceptions: these exceptions result from the malfunctioning of the underlying infrastructure that supports the WfMSs. These exceptions include hardware errors such as computer system crashes, errors resulting from network partitioning problems, errors resulting from interaction with the Web, errors returned due to failures within the Object Request Broker (ORB) environment, etc. In the newborn infant transport workflow, an infrastructure exception can be caused by an error in the telecommunication media between the ambulance and NICU.
- Workflow exceptions: All exceptions form a hierarchy rooted at class Exception. Two basic groups of exceptions include system exceptions and user-defined exceptions. A variety of system exceptions identify a number of possible system-related

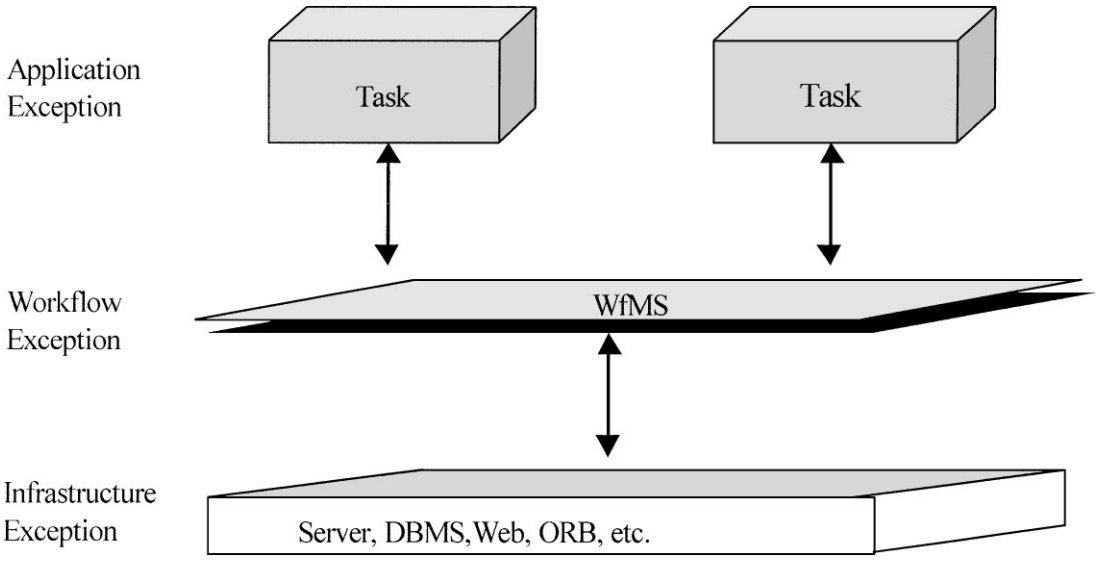

Figure 4. Three layered exception model.

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, etc. User-defined exceptions are specified by the workflow designer and identify possible application-dependent deviations in task realizations. Specific user-defined exceptions depend on the specific workflow application. An exception handler is a description of action(s) that workflow runtime component(s), or possibly a workflow application, is going to perform in order to respond to the exception.

- 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 newborn infant transport workflow, an application exception can be caused by an error in health professionals assignment such as agents could not be found for the roles required.

3.7. Exception Detection

One of the important tasks in exception handling is to detect exceptional situations. The objective of exception detection is to capture deviations in the systems. Exception detection can be achieved by supervising the workflow system components' external inputs and outputs, and comparing their behavior with the specified behavior of the system. In this approach, the supervisor predicts a single likely behavior of the system and, if the observed behavior does not match the prediction, rolls back and creates a new prediction of the valid behavior. An exceptional situation is identified when the supervisor has explored all valid behaviors without matching the observed behavior, resulted from which an exception is raised. In the following, we are going to discuss several supervision techniques.

- Under-specified specification supervision: Under-specified specification often results in a non-deterministic system specification. Non-determinism in specification is advantageous because the specification writers can avoid stating irrelevant behavior as mandatory, freeing the software designer to choose a behavioral alternative that would yield a more desirable implementation. The major difficulty in supervising such systems is that the supervisor must account for all possible behaviors that are permissible under the non-determinism present in the specification.
- Hierarchical supervision: this type of supervisions is very natural due to the hierarchical construction of WfMSs. The hierarchical approach differs from common one-layer approaches in that supervision is split into several sub problems such as tracking the external behavior of the target system and detailed internal behavior checking.
- Time supervision: In time critical software, a correct output that is not produced within a specified response time interval may also constitute an exception. Automatic detection of time-related exceptions is achieved through tracking the state of a workflow as well as the elapsed times between specified request and response pairs. System's time behavior is derived directly from the workflow application's requirement specifications. Time supervision allows exceptions in a workflow system to be detected in real-time based the specification of its external behavior. This feature is of particular benefit because a WfMS often needs to integrate legacy systems, software component purchased from other vendors, and tasks developed by third parties.

3.8. Exception Handling

We have conducted several workflow projects such as modeling and development for real-world workflow applications (e.g., the statewide immunization tracking application [13]), and in using flexible transactions in multi-system telecommunication applications [14]. In the following, we formulate some of the essential requirements for exception handling based on our prior experiences and our understanding of the current state of workflow technology and its real-world or realistic applications [2, 15].

- Support specification for exception handling: We allow an application developer to specify various types of user-defined exceptions to catch anticipated failures or deviations.

- Support task-specific exception handling: Tasks are viewed as black boxes for a WfMS. The workflow enactment service doesn't have to be concerned with the realization of the tasks. The workflow tasks, in general, are more complex than database transactions, and represent a logical activity in the overall organizational workflow. It is therefore critical to be able to detect the exceptions returned by arbitrary tasks.
- Localize exceptions: The first attempt to handle exceptions is to isolate and mask them. This can prevent the problem affecting other parts of the system.
- Support exception handling by forward recovery: we give a general discussion about exception handling through workflow recovery.
- Support human-assisted exception handling: It is impossible to guarantee the success of exception handling mechanism due to the non-deterministic nature of exceptions. Therefore, the involvement of a human is critical for resolving erroneous conditions that could not be dealt with by the WfMS automatically. Thus, human involvement is a very important element for exception handling.

We adopt an exception handling mechanism that involves exception masking and propagation (see Fig. 5). As shown in Fig. 5, a task has four states, *initial*, *execute*, *fail*, and *complete* [16]. In the *execute* state, the task is actually being executed. When the execution finishes, the task either enters *fail* state or *complete* state. The task can throw many exceptions. An exception is masked if a local exception handler can handle it. Otherwise, the exception will be propagated. There are two types of exception propagation: *structural exception propagation* and *knowledge based exception propagation*. The structural exception propagation follows the control (or calling) structure of the workflow inter-task dependencies enforcement. The knowledge based exception propagation can directly propagate exceptions to knowledgeable workflow components such as a CBR based exception handling component and human agents.

Exception masking prevents an exception in one part of the workflow systems from affecting other parts of the systems. Our intention of masking exceptions tries to capture an exception as close to its point of occurrence as possible, and a suitable handler at the same level of that point will first handle this exception. Exception masking includes corrective actions such as ignore, warnings, retry, procedural actions, workflow recovery, workflow modifications, suspend/stop, and so on. In the following, two categories of exception-masking techniques (hierarchical and grouping) are discussed that are used to organize the corrective actions.

- Hierarchical exception masking: If a component depends on lower-level components to correctly provide its service, then an exception of a certain type at a lower level of abstraction can result in an exception

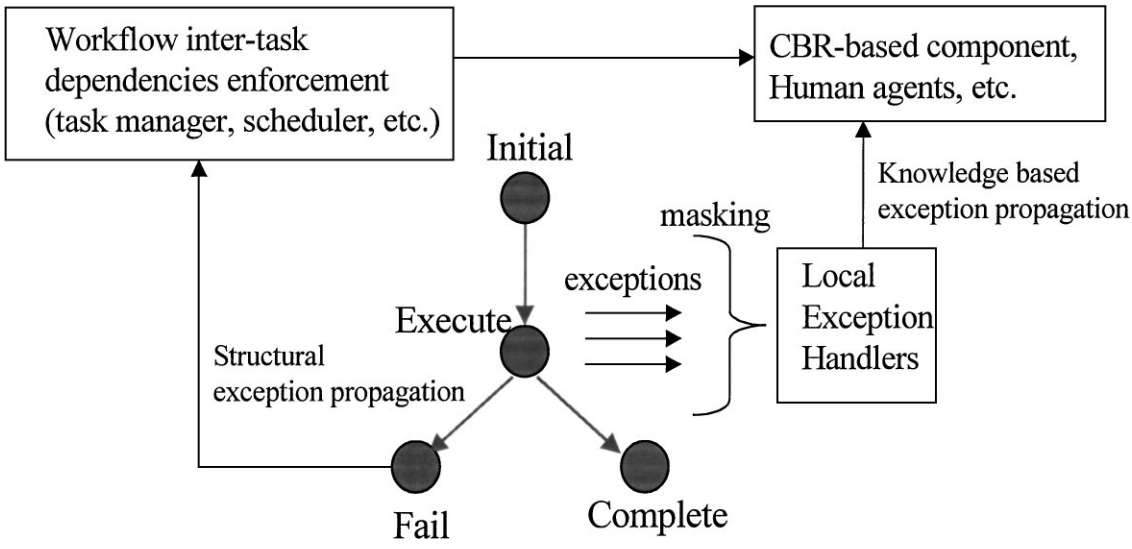

Figure 5. Exception masking and propagation in defeasible workflow.

of a different type at a higher level of abstraction. Exception propagation among components situated at different abstraction levels of the hierarchy can be a complex phenomenon. For example, if WfMS needs services provided by infrastructure, such as ORB, Internet, with arbitrary exception semantics, WfMS will likely have arbitrary exception semantics, unless it has some means to check the correctness of the results provided by infrastructure. In such hierarchical systems exception handling provides a convenient way to propagate information about exception detection across abstraction levels and to mask low-level exceptions from higher-level components.

- Group exception masking: One way to ensure that a service remains available to clients despite server failures is to implement the service by a group of redundant, physically independent servers, so that if some of these fail, the remaining ones can provide the service. Group masking can mask the exceptions of an individual member, whenever the group responds as specified to service requestors despite the individual's exceptional situation. With group masking, individual member exceptions are entirely hidden from service requestors by the group management mechanisms. Group masking uses redundant information to deliverer the correct service.

3.8.1. Exception Handling Through Workflow Recovery. Like [17], we believe that ignore, retry, backward recovery are good exception handling candidates that can be used in different situations. For example, retry is not applicable if the workflow applications cannot be retried. Serious exceptional situations cannot be ignored. Furthermore, 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 ACID properties of transactions [18]. 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 the error 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 based exception handling mechanism that would semantically undo, or compensate a partially failed task.

3.8.2. Exception Handling Through Dynamic Workflow Changes. Due to foreseen and unforeseen situations, i.e., exceptions, pre-defined workflows may need to be modified to adapt to such exceptional events. However, the resulted workflows must be correct eventually, i.e., the workflow graph should exist after corrective actions are performed. Possible modifications to workflows can be at instance level or at model level. The ways of modifications that will result in correct workflows are as follows.

- Modifying JECA rules: It can modify any part in a JECA rule. To ensure a workflow graph exists in the resulted workflow, the enabling relationships should be kept.
- Inserting JECA rules: There are two ways of insertions. One way is to insert the rules parallel to an existing rule. If the inserted rules have same enabling relationship as the existing rule, the resulted workflow remains correct. The other way is to insert a new rule r between two existing rules r_1 and r_2 that r_1 enables r_2. The r_1 and r_2 must be modified such that r_1 enables r, and r enables r_2.
- Removing JECA rules: If the removal of a rule does not modify the enabling relationship, the resulted workflow remains correct. Otherwise, if rule r is removed from two rules r_1 and r_2 that that r_1 enables r, and r enables r_2, rules r_1 and r_2 need to be modified such that rules r_1 enables r_2.
- Commutative change: For example, if enabling relationship between rules r_1 and r_2 is commutative, r_1 enables r_2 can be changed to r_2 enables r_1.
- Combination of the above.

3.8.3. User Exception Support. Users can anticipate and provide solutions to deal with certain exceptional situations. To support 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 Fig. 6). Exception sources include errors, faults, failures, and other exceptional situations. Like [19], we view constraints as exception sources too. That is, when constraints are broken, exceptions may be raised. By providing such a mapping mechanism, workflow application designers can decide which

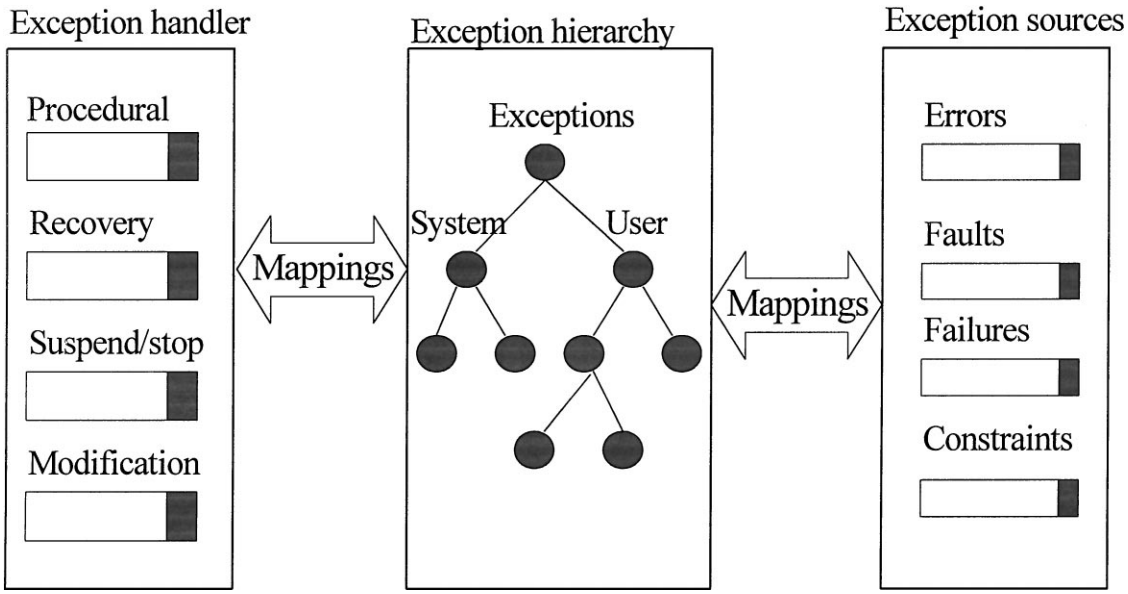

Figure 6. Mappings between exceptions, exception sources, and exception handlers.

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.

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 play 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.

Figure 7 shows an example of user exception-handling scenario that involves taskAssignException exception. This exception will be declared as a 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 users 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.

3.8.4. Knowledge Based Exception Handling. Exceptional situations are usually very complicated. A knowledge-based approach is a good candidate in dealing with such complicated situations. It can help workflow designers and participants better manage the exceptions that can occur during the enactment of a workflow by capturing and managing the knowledge about what types of exceptions can occur in workflows, how these exceptions can be detected, and how they can be resolved. Exception knowledge bases should be generic and reusable. This approach should allow the users to navigate through that knowledge base to find support for his decision on how to handle a certain exception. An explanatory module could also be

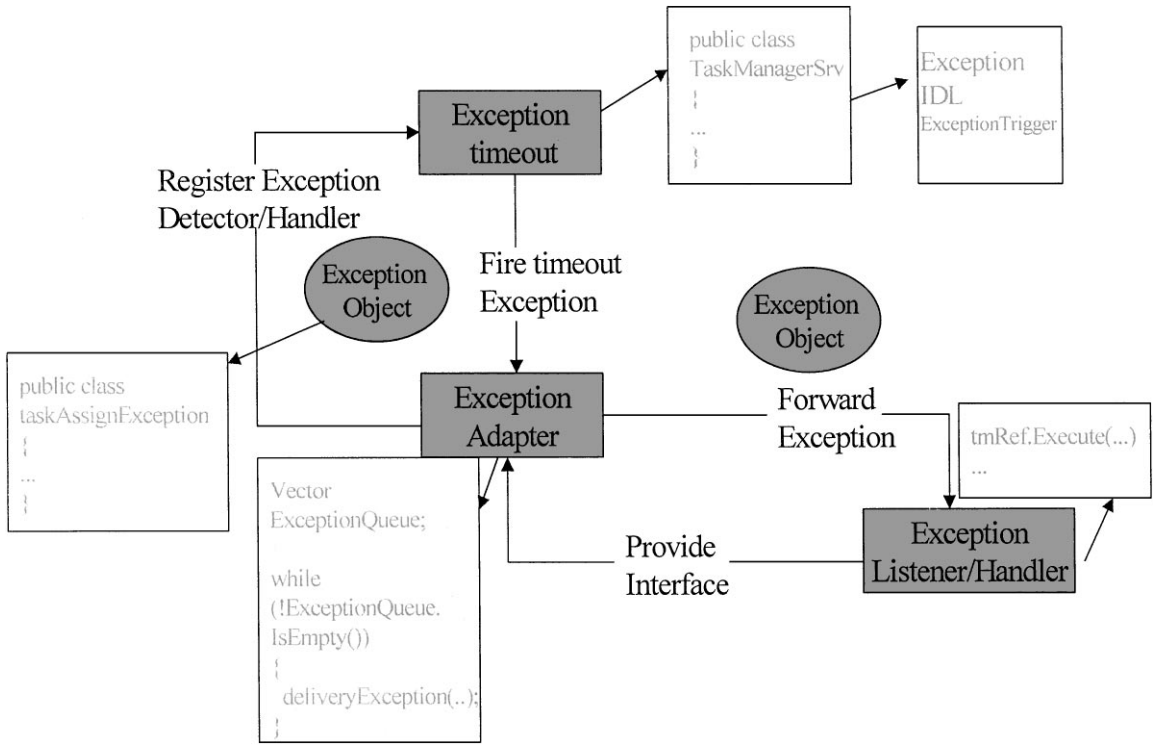

Figure 7. User exception handling scenario.

incorporated into the knowledge systems to explain the exceptional situation and solutions. We proposed a CBR-based exception handling mechanism that addresses these requirements.

4. CBR-Based Exception Handling

In this section, we pay more attention on dealing with uncertainties, experience acquisition, and supporting human involvement in exception handling through a CBR-based exception handling mechanism. In the infant transport application, when an exceptional situation is discovered the personnel on board should make decisions, assisted by the process automation mechanism, as to which tasks should be taken. A CBR-based exception handling mechanism with integrated human-computer processes is used to facilitate such decision-makings in handling exceptions. When the case-based reasoning component is notified about the exceptional situation of the infant, similar cases stored in the case repository will be retrieved and analyzed. The result from the analysis will be reused to facilitate the decision making process in exception handling.

4.1. Case Resolution

The CBR based approach models how reuse of stored experiences contributes to expertise [5]. 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 usually consists of three parts: problem, solution, and effect as illustrated in Fig. 8. Problems will be obtained through user or system input. There are solution candidates for those problems. The selection of the output solution will be based on the analysis of the effects of those solutions.

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 also be incorporated into the case repository. Effects of the exception handler candidates on the workflow system and applications will be evaluated.

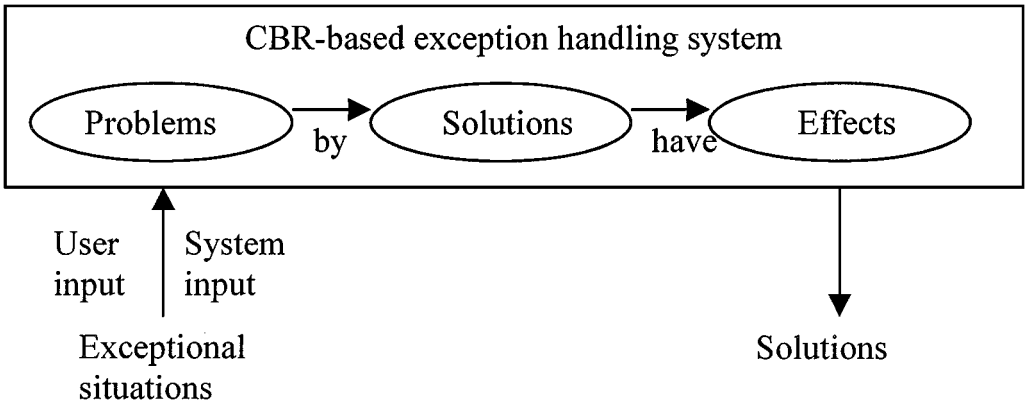

Figure 8. Exception handling via CBR based mechanism.

Thus, when the exception is handled, necessary modifications to the workflow systems or applications may be made. There are four steps in the exception handling by exception handler that involve case-based reasoning:

- Retrieval of the most similar cases to the identified exceptional situation,
- Analysis of the solution from the most similar cases,
- Adaptation of the most similar cases, and
- Updating of the system by adding the verified solution to the case repository.

4.2. Case-Based Reasoning Architecture

As shown in Fig. 9, 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: it 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: manages the concept in multiple domains.
- Similarity Measure component: provides similarity measure algorithms during case retrieval. Cases are often heterogeneous. Different cases need different similarity measure algorithm. This is achieved through separation of this component from retrieval component.

When exceptions occur, the abstractor identifies the exceptional situation via input (system input or user input). It extracts information from input. Output of abstractor is fed to retrieval component. It 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 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 interact in deriving acceptable solutions. During the interaction, the explainer helps users to understand the cases.

4.3. Ontology-Based Case Management

Since this case-based reasoning system needs to apply in various domains (such as healthcare, telecommunication, finance, etc.), we use ontology to describe the concepts used in various domains. Figure 10 shows an example of a high level ontology for disorders in medical intensive care we developed for the healthcare

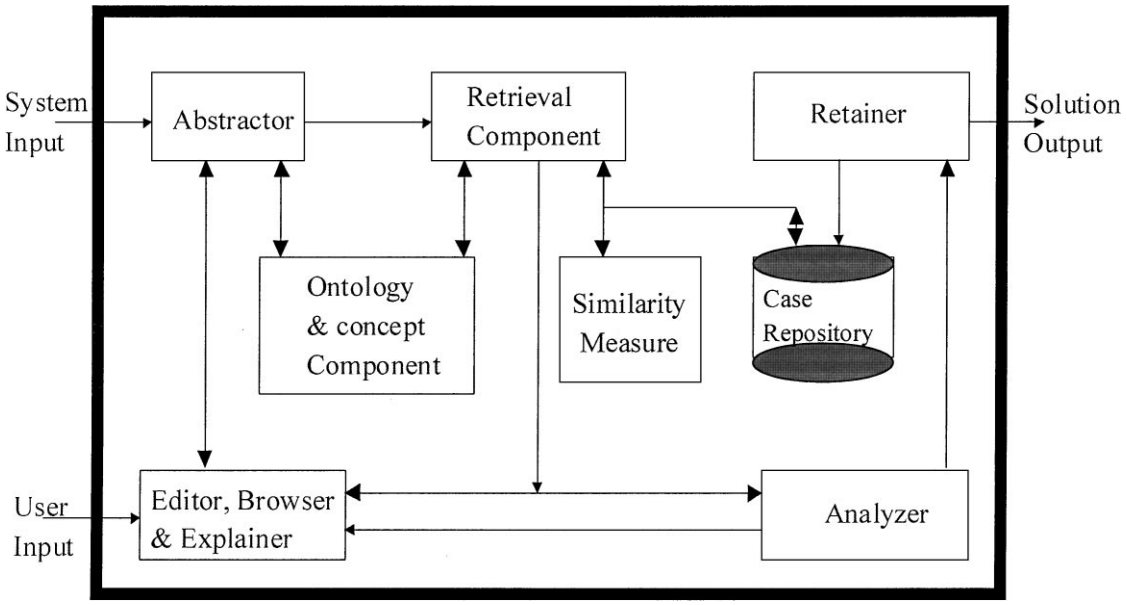

Figure 9. Case-based reasoning architecture in defeasible workflow.

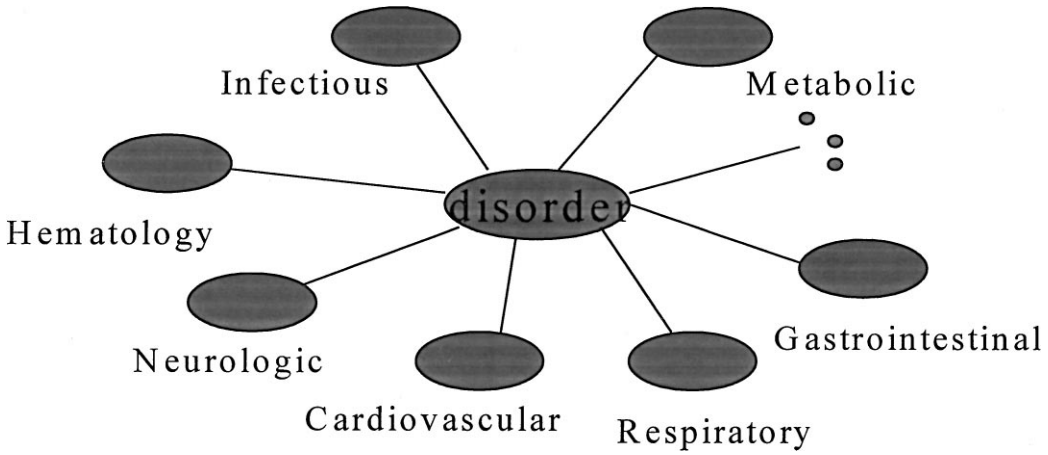

Figure 10. A high level ontology for disorders in medical intensive care.

workflow application. 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 this paper, we acquire new cases by learning through exceptions. That is, we create new cases when exceptions occur. We adopt an ontology-based case management:

- Abstraction: we extract information from exceptional situations and represent them as cases via ontology.
- Retrieval: we use ontology to organize cases. The retrieval of similar cases is based on ontology-based similarity measurement.
- Adaptation: adaptation increases the usability of existing cases.

4.3.1. Case Abstraction. Cases are derived from exception instances. An instance of the class Exception, or one of its subclasses, represents an exception. The exception instance or object should carry information from the point at which exception occurs to the

component that detects it. Due to exception masking and propagation, an exception object may pass among different components situated at the same or at different layers. During the propagation, information contained in the exception object may increase to record the handling history.

That is, an exception object may change during the propagation. An instance of an exception can be serialized when necessary to achieve a persistent exception object. In situations when case-based reasoning component is called to derive a handler for an exception, the information contained in that exception object is used in case-based reasoning to identify the exceptional situation.

Case abstraction involves two processes: determination of facts about exceptional situations and extraction of texts that can be used to summarize an exception based on ontology. The goal of extraction is to identify exceptional situations and extract index terms that will go into case repository. Rather than trying to determine specific facts, the goal for user input or text summarization is to extract a summary of an exceptional situation. The abstraction is used to represent the exceptional situations for search purposes or as a way to enhance the ability to browse a case repository. It is worth noticing that the abstraction usually is not necessarily complete. It may be just a partial description about the exceptional situation.

4.3.2. 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 can be achieved by computing the nearest neighborhood function of the quantified degrees of those semantic similarities (e.g., risk factors, age) between cases and structural similarities (e.g., AND, OR building blocks) between workflow specifications. To conduct a similarity based case retrieval, the similarities should be computed between targeted case (new situation) and old cases for three components: problem, context and resolutions. Each component may have its attributes that are application dependent. They are weighted according to the similarity measure algorithms used. The quantified value of similarity about each attribute between two cases is based on the ontology, for example, shown in Fig. 10. Such similarity measure setting up information is stored in the similarity measure, and ontology and concept components (see Fig. 9).

For example, there is a new situation that a newborn infant Mike Clinton needs initial intravenous maintenance. His age is about 15 days. To derive the actual solution, similar cases are retrieved. In this case, there are two attributes that can describe the exceptional situation, age and risk factor. Suppose there are two similar cases retrieved as shown in Fig. 11. Case a is retrieved because the new situation is described by the same term as case A—intravenous maintenance. Case B is similar to the new situation because the heart murmur is a kind of cardiovascular risk factor. Both intravenous maintenance and heart murmur are related

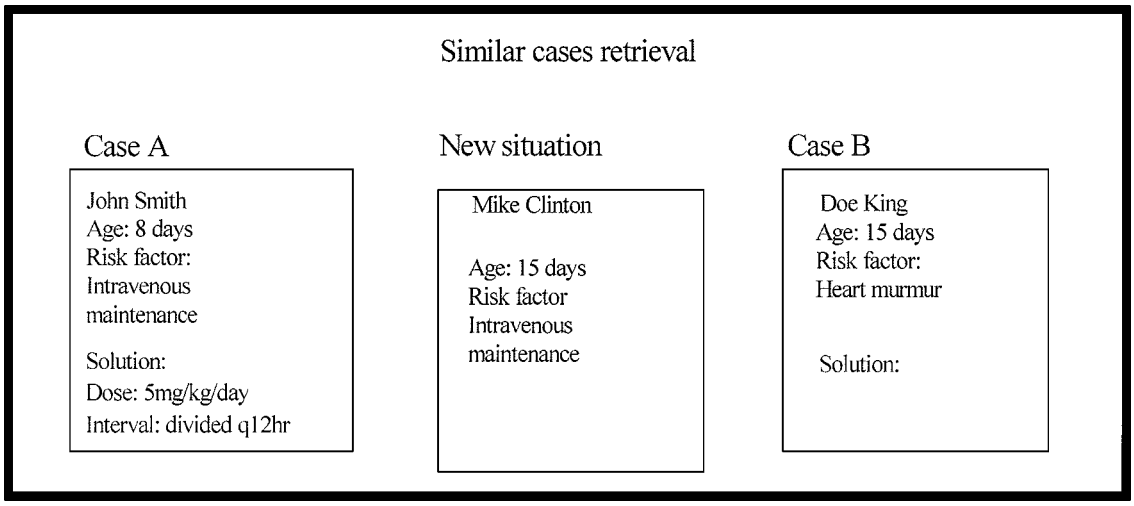

Figure 11. Example of two retrieved similar cases.

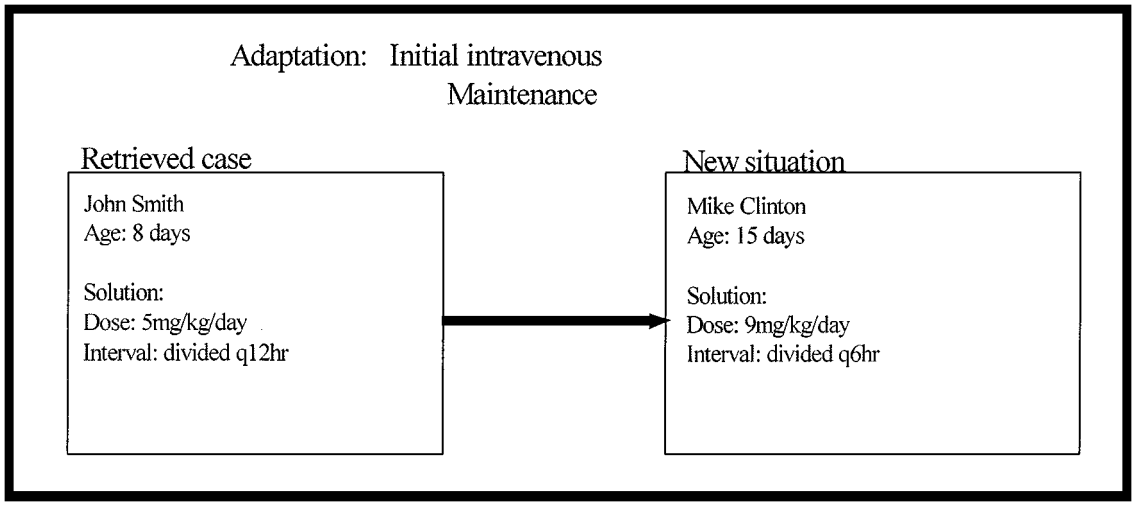

Figure 12. Example of parameterized adaptation.

to the cardiovascular concept. In determining which case is more similar, a smaller weight is put on age (weight 1) than on the intravenous maintenance (50). That is, the risk factor is more important than age if the age difference is less than 2 weeks. By using the one level ontology tree (see Fig. 10), the problem difference between case A and new situation is 0, while the difference between case B and new situation is at least 1 (because there is at least one level concept difference in the ontology tree). The similarity measure result between case A and new situation is $36((15-9)^2*1+0*1)$. The similarity measure result between case b and new situation is $50((15-15)^2*1+1*50)$. Thus, case A is the most similar case.

4.3.3. Case Adaptation. There are two main approaches to realize 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 [5]: (1) reuse the past case solution (transformational reuse), and (2) reuse the past methods that constructed the solution (derivational reuse).

There might be combinations of the above two approaches. However, a case can be used without any modification, 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. An example for parameterized modification is a physician might change the dose of a drug or other solutions according to the situations of a baby such as age or weight (see Fig. 12). Modification of a case by substitution usually results from the fact that the case or part of the case is not applicable in new situations or not available. In the healthcare domain, a physician usually can prescribe different drugs to patients with similar symptoms according to either patients' demand or situations of the drugs market (for example, a new, more effective drug has come into the market). Figure 13 shows another example for adaptation (modification) by substitution.

4.4. Integration of Case Management

There are two approaches for the integration of the CBR-based system into a WfMS. One is to integrate it as a built-in component. In this approach, an exception can be propagated to the CBR-based component through knowledge based exception propagation (See Fig. 5). Since CBR-based system usually does not participate the workflow inter-task dependencies enforcement, structural exception propagation that relies on control structures can also propagate the exception to CBR-based system usually as the last resort to handle exceptions. The other approach is to

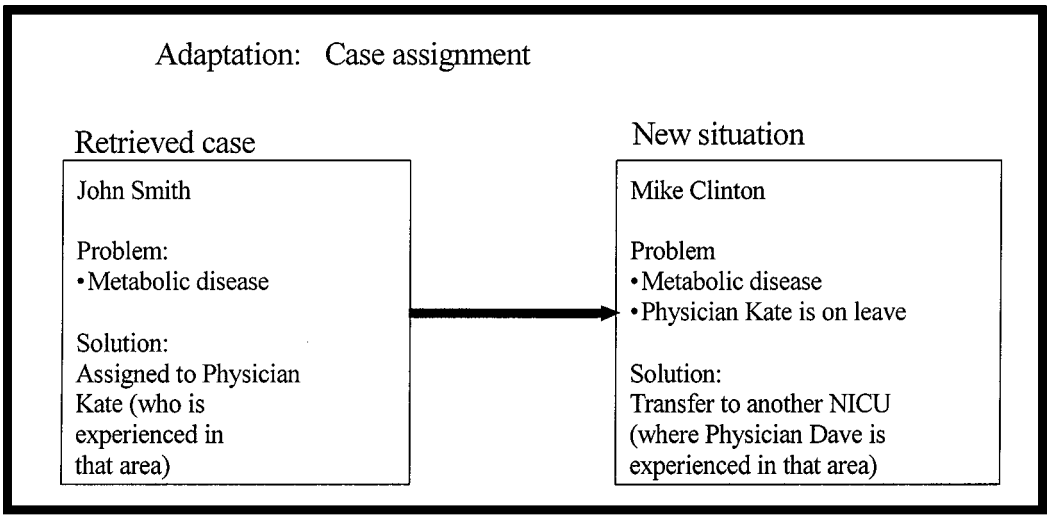

Figure 13. Example of adaptation by substitution.

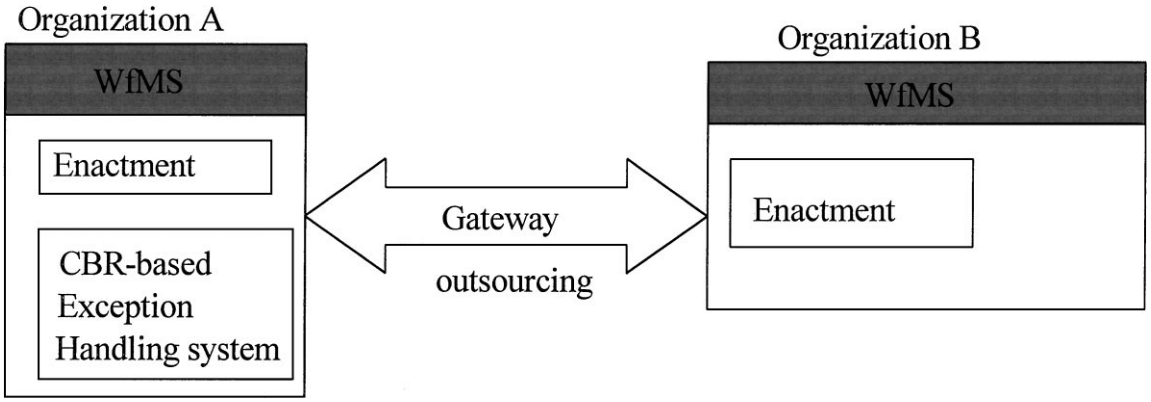

Figure 14. Cooperation between WfMSs in two different organizations.

outsource the CBR-based system in the context of inter-organizational workflows (See Fig. 14). Assume the CBR-based system has been integrated using the first approach in a workflow system. The solution of outsourcing the CBR-based system to other workflow systems becomes the cooperation between two workflow systems.

When an exception occurs in the organization B, organization B may want to document the exception handling process. The enactment component of WfMS in organization B needs to acquire a reference to the service requestor of the CBR-based service of organization A. Such a handshaking is accomplished through the inter-operable gateway as shown in Fig. 13. This gateway specifies how a service requestor can be obtained, how a service can be activated once service requestor is obtained and how the required acknowledgement is to be returned. An example of such a gateway is jFlow [20], a proposal admitted by OMG to meet the request for workflow interoperability. A more complicated gateway is proposed in [21]. The requestor is responsible for identifying the correct CBR-based service process. The process manager who acts as a process factory creates the process. The Requester is associated with the process when it is created and will receive states change notifications from the process.

5. Related Work

The importance of incorporating the exception handling mechanism into WfMSs has been identified by researchers and projects (e.g., METEOR [16], WAMO

[17]). The role of exceptions in office information systems has been discussed at length in [22]. The author presents a theoretical basis, based on Petri-Net, for dealing with different types of exceptions. This taxonomy presented is purely driven by organizational semantics rather that being driven by a workflow process model. [23] reports several works in workflow exception handling, which includes using ECA-rules to model expected exceptions [24], a general discussion about exceptions in systems based on objected oriented databases [25]. [23] also reports taxonomy for exceptions in workflow systems. That exception taxonomy can be reused in our CBR based exception-handling system to help measure the similarities among cases during case retrieval and analysis. Since exceptional situations are often very complicated, knowledge based systems are good candidates in such situations.

An exception is usually raised to signal exceptional situations like errors and failures. Error handling in database systems has typically been achieved by aborting transactions that result in an error [26]. In [27] a failure handling mechanism uses a combination of programming language concepts and transaction processing techniques. However, aborting or canceling a workflow task would not always be appropriate or necessary in a workflow environment. Tasks could encapsulate diverse operations unlike a database transaction; the nature of the business process could be forgiven to some errors thereby not requiring an undo operation. Therefore, the error handling semantics of traditional transactional processing systems are too rigid for workflow systems.

"Ad-hoc" workflows [28, 29] solve problems case by case. Our approach proposed in this paper support more than "ad-hoc" workflows. In the first place, we use a context-dependent approach to support adaptive exception handling. We support "ad-hoc" workflows in the sense when exceptions occur, non-standard exception handlers are not derivable, CBR-based approach is used as the last resort. In addition to solving problems as in ad-hoc workflows, the CBR-based exception handling system collects exception-handling cases, derives exception-handling patterns from the experiences captured in exception handling, and tries to reuse the prior gained exception handling experiences in the future.

6. Conclusions

Exceptional situations can be very complicated phenomena. Fully automated exception handling processes generally are not possible [30]. Mechanisms based only on "ad-hoc" exception handling are not acceptable either. An integrated human-computer exception handling mechanism should be present to support a systematic way of exception masking and propagation.

In this paper, we have given a three-dimensional perspective of exceptions that gives directions to build exception-aware workflow systems. We have formulated a defeasible workflow based exception handling approach. We identified the requirements for dealing with exceptions in heterogeneous workflow environments. Our exception handling mechanism involves mapping heterogeneous infrastructure and application specific exceptions into workflow exceptions. Ignore, task retries, alternate tasks, workflow recovery, workflow modifications, and other exception handlers have been integrated with the exception model to provide a complete solution that involves masking and propagation for dealing with exceptions encountered during workflow enactment. In addition, we propose a CBR-based system, enhanced by ontology based knowledge management, to learn from experience to provide a system assisted decision-making method to facilitate the understanding of exceptions and derivation of acceptable exception handlers. This involves reusing the experiences captured in prior exception handling cases.

Acknowledgments

Special thanks goes to Tarcisio Lima for providing background materials in developing ontology. We also have held discussions with our METOER team members Jorge Cardoso and Zhonqiao Li, etc. This work is partially supported by the NIST Advanced Technology Program in Healthcare Information Infrastructure Technology (under the HIIT contract, number 70NANB5H1011) in partnership with Healthcare Open Systems and Trials (HOST) consortium.

References

1. D. Georgakopoulos, M. Hornick, and A. Sheth, "An overview of workflow management: From process modeling to workflow automation infrastructure," *Distributed and Parallel Databases*, vol. 3, no. 2, pp. 119–154, 1995.
2. A. Sheth, D. Georgakopoulos, S. Joosten, M. Rusinkiewicz, W. Scacchi, J. Wileden, and A. Wolf, "Report from the NSF workshop on workflow and process automation in information systems," Technical Report, University of Georgia, UGA-CS-TR-96-003, 1996.
3. S. Jablonski and C. Bussler, *Workflow Management: Modeling Concepts, Architecture and Implementation*, International Thomson Publishing, 1996.

4. A. Cichocki, A. Helal, M. Rusinkiewicz, and D. Woelk, *Workflow and Process Automation: Concepts and Technology*, Kluwer Academic Publishers, 1997. ISBN 0-7923-8099-1.
5. A. Aamodt, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," *Artificial Intelligence Communications*, vol. 7, no. 1, IOS Press, 1994.
6. P. Meier and J. Paton, *Clinical Decision Making in Neonatal Intensive Care*, Grune & Stratton: Orlando, Florida, 1984.
7. V. Marek and M. Truszczynski, *Non-Monotonic Logic, Context-Dependent Reasoning*, Springer-Verlag, 1993.
8. S. Ceri, P. Grefen, and G. Sanchez, "WIDE: A distributed architecture for workflow management," in *Proceedings of RIDE 1997*, Birmingham, UK, April 1997.
9. G. Kappel, S. Rausch-Schott, and W. Retschitzegger, *Coordination in Workflow Management Systems—A Rule-based Approach*, Springer, 1998. LNCS 1364.
10. E. Baralist, S. Ceri, and S. Paraboschi, "Improved rule analysis by means of triggering and activation graphs," in *Proc. of the Second Workshop on Rules in Database Systems*, Athens, Greece, September 1995, edited by T. Sellis, vol. LNCS 985, pp. 165–181.
11. N. Paton (ed.), *Active Rules in Database Systems*, Springer, 1999.
12. D. Worah, A. Sheth, K. Kochut, and J. Miller, "An error handling framework for the ORBWork workflow enactment service of METEOR," Technical Report, Dept. of Computer Science, Univ. of Georgia, 1997.
13. 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.
14. M. Ansari, L. Ness, M. Rusinkiewicz, and A. Sheth, "Using flexible transactions to support multi-system telecommunication applications," in *Proc. of the 18th Intl. Conference on Very Large Data Bases*, Vancouver, Canada, August 1992, pp. 65–76.
15. A. Sheth and S. Joosten (eds.), in *Workshop on Workflow Management: Research, Technology, Products, Applications and Experiences*, Athens, GA, August 1996.
16. N. Krishnakumars and A. Sheth, "Managing heterogeneous multi-system tasks to support enterprise-wide operations," *Journal of Distributed and Parallel Database Systems*, vol. 3, no. 2, 1995.
17. J. Eder and W. Liebhart, "Contributions to exception handling in workflow systems," in *EDBT Workshop on Workflow Management Systems*, Valencia, Spain, 1998.
18. 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, 1997.
19. A. Borgida and T. Murata, "Tolerating exceptions in workflows: A unified framework for data and processes," in *Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration, WACC'99*, San Francisco, CA, February 22–25, 1999.
20. Joint Workflow Management Facility Revised Submission to BODTF RFP #2 Workflow Management Facility—CoCreate, Concentus, CSE, DAT, DEC, DSTC, EDS, FileNet, Fujitsu, Hitachi, Genesis, IBM, ICL, NIIIP, Oracle, Plexus, SNI, SSA, Xerox, *http://www.omg.org/cgi-bin/doc?bom/98-06-07*.
21. 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*, San Francisco, CA, Februrary 22–25, 1999.
22. H. Saastamoinen, "On the handling of exceptions in information systems," Ph.D. Thesis, University of Jyvaskyla, 1995.
23. M. Klein, C. Dellarocas, and A. Bernstein (eds.), *Online Proceedings of CSCW98 Workshop Towards Adaptive Workflow Systems*, Seattle, WA, 1998.
24. F. Casati, "A discussion on approaches to handling exceptions in workflows," in CSCW98, *Towards Adaptive Workflow Workshop*, Seattle, WA, 1998.
25. D. Chiu, K. Karlapalem, and Q. Li, "Exception handling with workflow evolution in ADOME-WFMS: A taxonomy and resolution techniques," in *CSCW98, Towards Adaptive Workflow Workshop*, Seattle, WA, 1998.
26. J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers: San Mateo, CA, 1993.
27. C. Hagen and G. Alonso, "Flexible exception handling in the OPERA process support system," in *18th International Conference on Distributed Computing Systems (ICDCS)*, Amsterdam, The Netherlands, May 1998.
28. M. Voorhoeve and W. Aalst, "Ad-hoc workflow: Problems and solutions," in *DEXA Workshop*, 1997.
29. H. Wedekind, "Specifying indefinite workflow functions in ad-hoc dialogs," in *DEXA Workshop*, 1997.
30. D. Strong and S. Miller, "Exceptions and exception handling in computerized information processes," *ACM Trans. Information System*, vol. 13, no. 2, 1995, pp. 206–233.

Zongwei Luo is a Ph.D. candidate in computer science in the University of Georgia. He works as a research assistant in the Large Scale Distributed Information System Lab. His research interests include e-commerce, enterprise application integration, work coordination and collaboration, and knowledge based systems. He is a student member of the IEEE and ACM. He received his BS, MS degrees in computer science and technology from Huazhong University of Science and Technology, China.

Amit Sheth directs the Large Scale Distributed Information Systems (LSDIS) Lab and is a full Professor in the Department of Computer Science. He also founded and serves as president of Infocosm Inc. (http://www.infocosm.com). His technical interests include enterprise integration, semantics in global information systems, digital libraries, e-commerce, and digital media. He received his BE in electrical and electronics engineering from the Birla Institute of Technology and Science, Phiani, India, and his MS and Ph.D. in computer and information technology from Ohio State University. He is a member of the IEEE and ACM.

Krys Koehut is an Associate Professor of Computer Science at the University of Georgia. His research focus on Database systems, computer systems for genome mapping, user interfaces, Computational Genetics. Dr. Kochut obtained his Ph.D. from Louisiana State University in 1987. His recent course instructions include Compilers, Software Engineering, and Advanced Software Engineering.

John Miller is an Associate Professor and the Graduate Coordinator in the Department of Computer Science at the University of Georgia. His research interests include Database Systems, Simulation and Workflow as well as Parallel and Distributed Systems. Dr. Miller received the BS degree in Applied Mathematics from Northwestern University in 1980, and the MS and Ph.D. in Information and Computer Science from the Georgia Institute of Technology in 1982 and 1986, respectively. During his undergraduate education, he worked as a programmer at the Princeton Plasma Physics Laboratory. Dr. Miller is the author of over 60 technical papers in the areas of Database, Simulation and Workflow. He has been active in the organizational structures of research conferences in all these three areas. He has also been a Guest Editor for the International Journal in Computer Simulation as well as IEEE Potentials.