INTERNATIONAL JOURNAL OF INTELLIGENT CONTROL AND SYSTEMS
VOL. 12, NO. 1, MARCH 2007, 2-14

Representing and Handling Workflow Exceptions in Highly Dynamic Healthcare Environments

Xiping Song, Minmin Han, and Thomas Thiery

Abstract **—Medical informatics researchers have studied how to use software technologies to provide decision support for using evidence-based medical procedures. For many of these efforts, dynamically managing the exceptions in the workflows is a key issue since the medical workflows must cope with a wide variety of patient medical situation changes as well as those of the healthcare environments. This paper presents an analysis of past research in managing medical workflow exceptions, and proposes future research that would benefit the medical applications. The paper is focused on three topics: representing, handling and analyzing exceptions. Based upon our analysis, we believe that techniques for verifying exception management models and for handling dynamic exceptions through dynamic workflow re-composition and re-scheduling should be useful and possibly essential for developing large scale, practical medical workflow systems.**

Index Terms**—Medical workflows, Exceptions, Exception Management.**

1. INTRODUCTION

A **workflow management system** is a software system that provides workflow definition and interpretation (i.e., workflow engine) mechanisms to support the workflow execution, possibly by integrating with other software applications. Workflow interpretation may invoke other software applications and guide human collaborations [20]. The workflow systems have been successfully applied to handle the routine business applications such as purchasing approval processes to improve the productivity and quality of the business operations.

With similar motivations, many medical informatics and software researchers have extensively researched to define, analyze and semi-automate medical workflows. Clinicians have developed and published a very large number of evidence-based medical procedures to disseminate and standardize the best practices. Such procedures are then used as the starting points for the researchers to develop and experiment with automated or semi-automated medical workflows systems. The paper [37] provides a classification of those medical workflows.

Xiping Song is with the Siemens Corporate Research Inc., 755 College Road East, Princeton, NJ 08540 USA (phone: 609-734-3362; fax: 609-734-6565; e-mail: xiping.song@ siemens.com).

Minmin Han is with the Computer Science Department, Lehigh University, 19, Memorial Dr. West, Bethlehem, PA 18015, USA (e-mail: mih9@lehigh.edu)

Thomas Thiery is with Technical University of Munich, Germany. (e-mail: thiery@mytum.de)

Experimented medical workflows include: diagnosis workflow management system [5], treatment/therapy workflow management system [13][35], hospital administration workflow management system [12][34], etc. Support for workflow has been largely provided by means of electronic reminders, alerts and work items (work list) for the medical staff.

It is well recognized that the medical workflow is the most dynamic workflow for which the workflow system is applied to support. The dynamism is uniquely caused by the changing patient state, unpredictability of the treatment outcome, special health conditions for each individual patient, the prioritization changes (e.g., emergency situations), etc. Because of all of those, managing exceptions are common occurrences in the daily life of medical professionals [27][41]. In addition, past research and experiments have also shown that effective management of exceptions in a workflow management system is crucial for its success [17].

A formal definition of exceptions might not be possible and may offer little practical value since, as this paper indicates, identification of an exception can change over the time as well. This paper only offers an informal definition of the exceptions. Usually, a major part of a workflow definition defines the "normal" behaviors in response to anticipated events and/or results. Briefly, an ***exception*** is an "abnormal" condition that is not expected by "normal" workflow. However, as we will discuss later, exceptions in medical workflows may cover a wide variety of events, *not limited to medical emergencies or errors*, depending on the application context and workflow design decisions.

Exception management includes exception definitions and defined procedures to response to the defined exceptions. Such procedures can be carried out by automated computer systems and human activities. Deciding whether an event is "abnormal" can be quite subjective. Workflow system designers can choose to use exception management to manage some system behaviors or events which might not be so "abnormal" (e.g., missing appointments might not be so abnormal in medical domain).

We believe that dynamically managing exceptions is unavoidable for any complex medical workflow systems for at least the following reasons:

- Medical staff will often have to first react to unexpected events with the patient, possibly not with all desired information, and then will update the tracking records to reflect those changes later.

- Patient conditions and treatment outcome are always unpredictable to certain extent. The workflow and its execution schedule must be adjusted quickly to cope with this unpredictability. The adjustment for one patient can have ripple effects on the workflows and their schedules for other patients.

- Physicians often want to "customize" (though such customization is often performed very dynamically and implicitly) the "normal" flow [36] to best care the patients within the given environments (e.g., medical devices, schedules, patients' insurance)

- Many medical systems to which a workflow system integrates may raise exceptions that need to be handled. For example, the medical guidelines supported by GLIF3 [33], EON [40], and PROForma [39], include exceptions.

Studying exceptions in medical workflows and exploring how such exceptions can be handled promptly, appropriately and smoothly within the workflow systems should improve healthcare quality and efficiency.

For example, an emergency room diagnosis workflow for acute abdominal pain [11] can be summarized as: physician takes the patient's history; then the physician performs a physical exam; and then the patient needs to take a set of lab and imaging exams. However, "abnormal" situations can happen to this simple workflow. For example, a patient may crash during any of the three diagnosis activities. If this occurs, instead of continuing the workflow, the physician needs to perform immediate emergency treatment to save patient's life. One additional exception would be one lab result may be delayed for hours so the physician may have to check with the patient's current condition, to see if some alternative checks may be used instead of waiting since the patient may need immediate treatment. As this example illustrates, managing exception situations is common for healthcare professionals.

We categorize the exceptions in Table 1 to illustrate a wide variety of exceptions in medical workflows, including both expected and unexpected exceptions. The expected exceptions are those that, based upon their existing medical experiences workflow designers know, may happen with certain low probability. The unexpected exceptions are those that are completely unknown within existing experience and research, and thus cannot be defined with details. We used the categorization based on predictability and exception source, as many other researchers [2][10][14][17][18][30][36] did. This categorization helps us understand what exception managing capabilities would be useful for medical workflow systems.

Table 1 Exception Categories

Expected Exceptions: those exceptions are expected to happen, but less likely than the main branch of the workflow.	
Exception Source	**Example**
Workflow tasks	During a medication administration task, patient is allergic to drug "ABC" so cannot give "ABC" to patient
External applications	"NO_AVAILABLE_BED" exception from hospital bed management system
Data changes	Patient heart rate drops to 50/min, which often indicates a sudden adverse change of the patient's medical conditions
Temporal constraints	Blood test task is not done on time
Resource changes	The operating room is no longer available as planned.
Unexpected Exceptions	
Add a new therapy into treatment plan. The new therapy is a result of the medical research.	

This paper does not discuss those exceptions that are at system-level (e.g., network problems) or at software application-level (such as memory allocation error, data type conversion overflow). We believe those exceptions are not much related with medical workflow design and are often handled at the software application or at the operating system level.

Software application level exception management is a very active research topic in software engineering area. Most of this research is focused on the programming languages. Researchers provide new framework and algorithms to support exception handling for OOP languages (e.g. [7]) or analyzing/optimizing the exception handling performance (e.g. [32]). The exceptions in the software applications are limited to the errors either in the software application or in the system software. The handling is often limited to logging the errors and aborting the applications in a way to minimize their negative impacts on the business operations (e.g., fail-safe, rollback the committed data).

In medical workflow systems, the exceptions may not be errors like those in computer programs, but rather, as a part of expected business conditions. Thus they have to be accommodated instead of simply aborting the execution and logging it as an error as in other type of programs.

The goals of this paper are to: 1) summarize the important topics for managing exceptions in medical workflows; 2) survey the current state-of-the-art academic/industrial research results on these topics; and 3) point out further research that can yield practical results. We used this effort to start our research in the workflow exception management and dynamic workflow management areas.

Section 2 summarizes exception research areas. In section 3, we describe the current state-of-the-art in those areas. In section 4, we discuss future research.

2. RESEARCH AREAS

The research for managing exceptions in general workflow systems can be categorized into three major areas:

1) *Representing Exception Management:* How to intuitively, yet formally represent workflow exception managements, including exception properties and handling activities.

2) *Implementing and Executing the Handling of Exceptions:* How to systematically build exception handlings. How to propagate exceptions to the different levels of the handlings.

3) Analyzing Exceptions: Collect and process information of past workflow exceptions regarding their occurrences and impacts to aid the exception management development. How to define and ensure the correctness of the exception managements with respect to their syntax, execution, and semantics.

2.1 Representing Exception Management

Exception management often needs to be clearly represented both for computer execution (supporting medical workflow executions and rescheduling of the workflows) and user understanding (e.g., medical staff training, workflow design). Moreover, since medical professionals will be involved in developing the workflow definitions, the representation of the exception management must be understandable by a variety of users from different domains, not only IT, but also medical professionals.

We have identified the following aspects of exception management that would need to be represented. The following properties are listed by the importance of the properties:

- **The class of exceptions**: the type of the exception provides basic information about the exception. One sample type of exceptions is "TIME_OUT" which indicates that the exception of this type will be raised when some activity is not finished within a certain pre-defined time period.
- **The condition when the exception is raised:** It can be the failure/abort/timeout of an activity (e.g., the blood test is not finished within 2 hours), data thresholds (e.g., patient body temperature reaches 100F), or any combination of multiple conditions.
- **Actions for handling the exception:** the actions and their execution sequences for handling the exception. Exception handling can be as simple as skipping an activity, or as complicated as workflow evolution. Some examples of the handling actions include 1) changing a workflow definition while some of its instances have been partially executed and 2) changing the execution schedule of a workflow.
- **The occurring context of the exception**: The exception can be raised during the external applications, or an activity of a workflow. For example, an exception might be defined as only occurring during the "check history" activity of "acute abdominal pain diagnosis" workflow.
- **The receivers of the exception event**: The possible receivers are one or more running instances of workflow definitions, some external applications, or certain roles played by humans. For example, a receiver can be a treatment workflow instance for a patient or a physician who is a specialist for certain medical conditions. A receiver can also be a workflow scheduling application that adjusts the schedules for executing the workflow instances that might be impacted by the exception.

It is not necessary for an exception definition to contain all these properties. For example, if the publish-subscribe pattern is used as the exception handling mechanism, the "receiver" property is no longer necessary. However, an exception management definition must have the first three aspects.

The above discussions are concerned only with one exception management. To effectively and clearly define a large number of (inter-related) exceptions for a non-trivial workflow system can be a much harder issue. The exceptions may have certain relationships, such as "is-a" or "is-part-of'. Exploring the use of those relationships to ease the exception management representation is also an interesting research topic. Addressing this topic should help understand the medical exceptions greatly.

2.2 Executing the Exception Handling

Exception handling needs not only representation but also needs to be implemented and executed in a timely and repeatable fashion. The following areas are related to this aspect:

- **Propagation**: to route an exception to the corresponding handler; It should define a routing strategy to ensure every exception will be handled.
- **Handling primitives:** a set of pre-defined actions as the primitives for implementing the handler (e.g., record an exception into an event log); Those primitives are identified through the analysis and componentization of the existing handlers.
- **Handling logic**: the algorithm for determining the concrete activities to handle exceptions. Examples include using explicitly defined handling logic for each possible exception or consulting knowledge-base for appropriate handling logic.

To propagate an exception is to forward the exception to the appropriate handler. One example is to use a flat table structure to map a certain type of exception to one exception handler. Another way is that exceptions can be

propagated like the exceptions in Java/C++ [19]: the exception will be forwarded to the higher control level if the lower level does not have a handler for it. There are other ways to propagate the exceptions. It is critical to ensure every specified exception be sent to the appropriate handler module. A generic handler on the highest level takes care of every unhandled exception. This can be, for example, a mechanism that involves human interaction for dynamic exception handling and for re-scheduling of the workflow executions.

Researchers have identified and defined some primitives for coding the exception handler. We have studied [17][18][29] and combine their work with our experience to summarize possible primitives in Table 2.

Table 2 Exception Handler Primitives

Maintaining workflow "normal" behavior under the exceptional conditions	
Ignore	Takes no action.
Record	Record to log (e.g., develop audit trail)
Notify	Inform a role/an actor/a group of actors /external applications.
Propagate	Route the exception to another handler.
Resource	Add more resource.
Controlling the execution and modifying the behaviors of one/more workflow instances	
Retry	Retry the current task for specified number of times.
Suspend	Pause the current task/process
New	Add a task/process
Modify	Modify a task/process
Remove	Remove the current task/process
Change Sequence	Change the task sequence in the current workflow
Terminate	Terminate one/more processes.
Change Resource Requirements	Assign the task to another actor or change other resource requirements /constraints.
Delay	Delay a task/process
Modifying workflow definition (evolution)	
Modifications	Add new tasks, remove tasks, and change sequence of tasks…

The simplest and static, exception handling mechanism is to define the exception handling actions for each exception before runtime. A more flexible and dynamic mechanism is to let users define/select the handling during runtime. A more automated, adaptable exception handling mechanism is to automatically decide how to handle the exceptions depending on the previous exception handling experience, knowledge, resource limitations and other considerations. This mechanism can alter the workflows and their execution schedules to handle the exceptions.

2.3 Analyzing Exceptions

Exception management in a workflow system needs to be analyzed and verified to ensure correctness with respect to its syntax, executability, semantics and completeness. The analysis includes three sub-areas:

- **Identify all concerned exceptions:** what exceptions need to be defined and implemented into the medical workflow systems.
- **Verify the exception management before runtime:** The verification can focus on both the syntax and executability. Further, it might verify whether exception management would violate any clinical protocols.
- **Check the effectiveness of the exception managements**: Check whether the defined exception management is sufficiently complete and comprehensive. For example, it can check if the exception managements will handle all emergency conditions effectively, with the support of the knowledge bases.

Also since exceptions are usually built incrementally, it is important to check the exception management models to ensure they will remain working as they are being built. Programming languages like Java simply verify if possible exceptions are thrown to the higher level or caught at this level. For medical workflows, we need to make sure not only all exceptions are handled, but also the handling will not cause problems, such as conflicting handlers (as more and more exception management is being added) or break certain medical practice rules. This is because the exception management in software is limited to error logging or other simple handling. However, for the medical workflow, the handling can be more complex and be an integral part of the medical application.

3. CURRENT STATE-OF-THE-ART

Research for exceptions in workflow management systems has been going on for about ten years. In this section, we discuss the current research status in each of the three areas.

3.1 Representing Exceptions

By reviewing existing literature, prototypes and products for exception management, we found two general approaches to represent exception management: 1) *embedded* in workflow process definition [6][16][17][42]; 2) *stand-alone* [8][9][21][25][30]. The embedded approach is to expand the workflow process definitions to include exception management. By this, it defines the occurring context of an exception clearly (i.e., a scoped portion of the workflow process definition). The stand-alone approach is to separate the exception management from workflow process definition. This approach must use the condition to define the occurring context which might not be always possible and usually hard to be defined easily and clearly.

Two examples of embedded exception definition are Business Process Modeling Language (BPML) [6] and WfMC's XML Process Definition Language (XPDL) [42]. Since they are similar, we describe the XPDL approach only.

In XPDL, exceptions are defined as a special type of transition between activities. Figure 1 shows a partial XPDL specification: the normal workflow is first to take X-Ray chest film. Then the physician checks the patient history. When the "XRayNotAvailable Exception" happens during task "XRay Chest Film", a "CT Chest Film" task is created and needs to be completed before the "CheckHistory" task. Note that arc from "XRayNotAvailable" to "CT Chest Film" is not a *conditional* branch, but an exception handling that might happen during the entire "XRay Chest Film" activity.

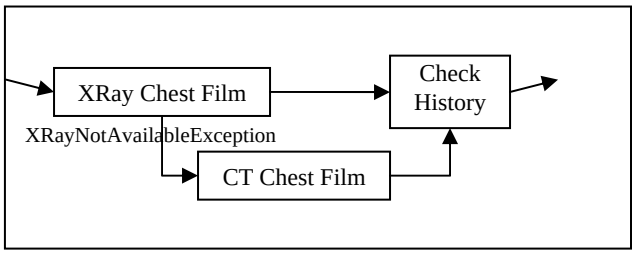


Figure 1 partial XPDL specification for XRAYNotAvalable Exception

Figure 2 shows the textual definition of Figure 1. The raise of an exception is treated as a special type of transition condition and the handling of the exception is defined as a branch caused by the exception.

The only difference for an exception transition and a normal transition in XPDL is that the former has a different type and the transition is labeled with the exception name. "CT Chest Film" is an exception handling task. However, the exception handler is scattered in the diagram and the textual definition together with the other "normal" transitions.

The "stand-alone" approach uses ECA (Event, Condition, Action) rules and knowledge base entries [8][21][26][30]. Klein et al. specify exceptions in a knowledge base [26]. Every exception is associated with a knowledge base entry that includes: an exception definition, a definition when the exception will become critical, a detection process, and a handling process. ECA rules include three components: event, condition and action, which are originally used in active databases [43].

The "stand-alone" approach uses ECA (Event, Condition, Action) rules and knowledge base entries [8][21][26][30]. Klein et al. specify exceptions in a knowledge base [26]. Every exception is associated with a knowledge base entry that includes: an exception definition, a definition when

```
…
<Transitions>
  …
 <Transition Id="br1" From="XRay Chest Film" To="Check History"/>
 <Transition Id="br2" From="XRay Chest Film" To="CT Chest Film">
     <Condition Type="EXCEPTION">
        XRayNotAvailableException
     </Condition>
 </ >
 <Transition Id="br3" From="CT Chest Film" To="Check History" />
  ...
</Transitions>
…
```

Figure 2 Textual Version of XPDL specification

the exception will become critical, a detection process, and a handling process. ECA rules include three components: event, condition and action, which are originally used in active databases [43].

Chimera-EXC [8] provides a detailed example of using ECA to specify exceptions. A trigger is defined for global exception or an exception for a specific process. It includes events (E), conditions (C), actions (A) and the event priority. If an event arrives and the conditions are satisfied, the actions will be executed. If more than one triggering event arrives, the corresponding actions are executed according to the event's priority (i.e., the order can be defined to indicate the priority).

```
define trigger XRayNotAvailable
   events              raise (XRayMachineBroken)
   condition           XRayChestFilm(X), X.status = "waiting",
                       DiagnosisCase(C), X.instanceOf(C)
   actions             cancelTask(X),    startTask(C,
"CTChestFilm")
   order 1
```

Figure 3 Chimera-EXC specification example

Figure 3 shows how to define the exception described in Figure 1 using Chimera-EXC. Once an external event "XRayMachineBroken" arrives and if a XRayChestFilm task is at the waiting state, a new task "CTChestFilm" is started instead.

An event of Chimera-EXC can be a data update event, a workflow generated event, a temporal event (time point or time interval) or an external event that is raised by a third party software application. Its conditions are set with context variables, which support actions such as setting data/context variables or modifying tasks.

An embedded exception specification can be easier to understand because it is within an application context where the number of involved exceptions is limited. However, just as [3][19] pointed out, embedding exception definition and handling into a process definition can

reduce its exception handling ability since the large number of exceptions will obscure the normal workflow process. One example is the "patient crash" exception, which may happen at anytime and thus it is related to all tasks in the workflow. Currently, as seen from the literature, the "stand-alone" exception management definitions provide more expressive power by allowing the use of complex conditions (including complex temporal conditions, e.g. within 2 hours of last medication). However, exception definitions separated from the "normal" workflow process definitions may be harder to understand since they are dislocated from their application context.

Strengths and Limitations:

- *The current approaches to exception representation provide enough expressiveness:* however, no one representation approach provides all the expressive power we would need; instead we need a combination of them. For example, Chimera-EXC provides support for complex temporal conditions required by exceptions in medical workflows but its action definition is descriptive instead of a declarative as in XPDL.

- *Understandability for various kinds of users is still lacking:* Different users are concerned with the representations at different definition levels and for different aspects. A set of views should be designed for different user roles. For example, physicians want to see an overview of the exceptions and their handlings embedded within the workflows while they also want to see a separate view of the details of exception properties when modifying the exception definitions. Probably, for patient education, patients would need a more friendly graphic presentation so that they can understand the relevant exceptions and handlings, likely even with the related cost, risks, and possible alternatives.

3.2 Handling Exceptions

The exception handling propagation mechanism of many workflow management systems is similar to the nested exception propagation mechanism in programming languages such as Java and C++ [19]. In using this mechanism, an exception is passed along the workflow call hierarchy to higher level handlers if the lower level module cannot handle it [8][10][24][29]. Alternatively, a workflow can subscribe to specific system events that should handle. In this case, there is no workflow call hierarchy for exception passing. The handling for an exception defined at the individual workflow level may not be defined or visible with the exception definition.

Figure 4 shows the propagation of an exception in the OPERA system [24] that is based upon the workflow call hierarchy: process p0 creates sub-process p1, which creates activity p2. When an exception event that is subscribed by p2, is raised during the execution of p2, and if p2 does not have a handler, the exception is propagated to its parent p1. if p1 also has no exception handler; again the exception is propagated to its parent p0. p0 has an exception handler for this exception and does the handling, for example, that could abort p2 and resume p1.

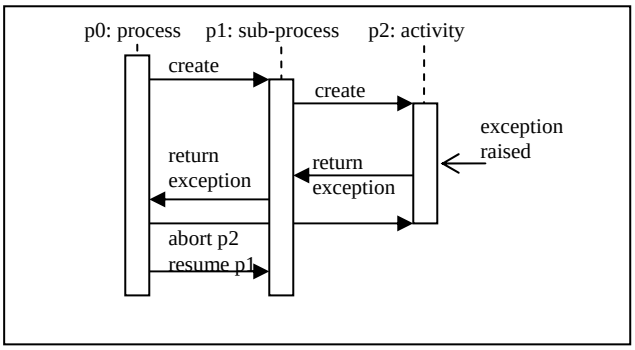


Figure 4 Propagation of Exception in OPERA

When an exception is propagated to the root of the workflow call hierarchy without getting caught by an exception handler, some workflow management systems have a specialized module to deal with this situation. In defensible workflow [30], when no handler can be found for an excepting, the system will try to solve the problem based upon past experience. When the system finds no similar past experience, human intervention will be required.

We consider this propagation mechanism usable for medical workflows. The exception handler hierarchy makes reuse of handler modules possible and the root default handler help make sure that any unhandled exception will be caught and handled eventually.

We have investigated a number of workflow management systems [9][10][17][21][30][36][37] to see how well the primitives in Table 2 had been implemented. As a result of our investigation, we found each of those approaches supports either a smaller or bigger set of the identified primitives. We found that any of those primitives is supported by some systems investigated, though they might be implemented in different ways. Thus, we believe that these primitives have been sufficiently supported by existing workflow management systems. Exception handling mechanisms include:

- **Pre-defined handling:** The actions for handling an exception are explicitly defined by the user before runtime [9][36]. ADOME-WFMS has system built-in modules to provide automatic pre-defined handling for some expected exceptions such as best candidate actor is not available [10].

- **Ad-hoc handling:** Users are allowed to specify or select the exception handling actions from the exception handling building blocks when an exception occurs [10]. This mechanism is mainly for unexpected exceptions.

- **Using an extended model to enable workflow recovery:** Extend the ACID (Atomicity, Consistency, Isolation, Durability) transaction model to support

automatic exception handling in workflow management systems [17][18][23][24][37].

We found that researchers proposed and developed algorithms to label tasks, and then to use backward recovery (partial rollback), forward recovery (partial rollback and use alternative route), or pre-defined compensation process according to the failed task's label and the structure of the process. The process state may end at the regular committed states, failed state or other extended states (such as compensating or aborting). Such mechanisms provide support for workflows to handle or adapt to the exception condition while still working towards accomplishing their goals. This is quite useful for building healthcare workflows to be adaptive, not just simply to report a failure.

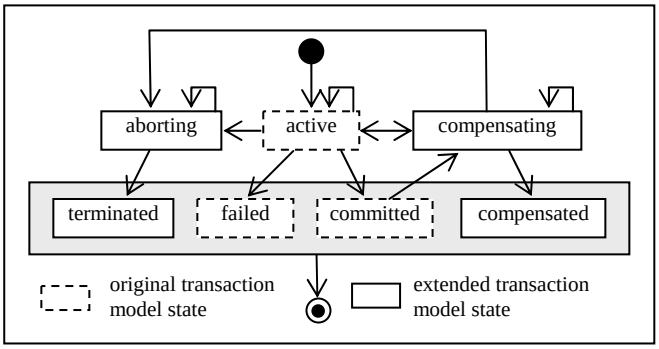


Figure 5 Extended Transaction Model [17]

One example of extended ACID transaction model based exception handling is from Eder et al. [17]. Figure 5 shows their extended transaction model: the arrows are possible transitions between process states. A process starts as active state when its first activity is started. If there is a failure or exception, it may enter "aborting" or "compensating" state. Besides the state "failed" and "committed", the final state of this process can be "terminated", or "compensated".

Eder et al. label the activities as vital (important activities) and non-vital (not so important activities). When a non-vital activity fails, "forward execution", i.e. to ignore the failed activity and continue the workflow, is used. When a vital activity fails, "backward recovery" and "forward recovery" (use alternative activities if possible) is used. The actual algorithm for backward recovery depends on the control structure of the failed activity (sequence, parallel, choice, or loop). The process' state is changed accordingly.

- **Knowledge-based handling:** To reuse the stored experience to handle exceptions. Klein [26] and Luo [29] discussed finding appropriate exception handling actions from a knowledge base. In defensible workflow, case-based reasoning (CBR) is used to analyze the case repository to find similar experience to handle exceptions [30]. Stored experience may be explored once an exception happens for quickly getting a solution. However, it is also desirable to analyze the stored experience off-line to improve exception handling for the future. More discussion for this topic is presented in section 3.3.

We believe that an extended ACID model is not suitable for treatment/diagnosis workflow exception handling because the rollback of medical treatment/diagnosis activities may be complex and context related (e.g. using alternative medicine for a patient depends on the patient's age, stage of disease, medical history and other information besides one workflow instance). Thus, we believe that it would be difficult to use the extended transaction model in treatment/diagnosis related medical workflows but possible for administration medical workflows. Knowledge-based handling can be very useful for medical workflows since medical knowledge grows quickly and this can help medical staff use the latest medical developments.

Strengths and Limitations:

- *Propagation mechanisms are satisfactory*. We think that medical workflow systems have no major special requirements for propagation. The current propagation mechanisms are capable of sending exceptions to an appropriate handler for medical workflows.
- *The identified exception handling actions are sufficient and broadly implemented.* We believe that the currently defined and implemented primitives are sufficient for both general workflows and medical workflows. However, the mechanisms for rescheduling and re-organizing the workflow instances are lacking.
- *Handling mechanisms can be made more intelligent.* We believe that the knowledge-based handling can benefit medical workflows because it can help the medical staff use the latest information and knowledge.

3.3 Analyzing Exceptions

Analyzing those exceptions that were unexpected, but occurred in the past, can help increase the capability of handling them in the future, while analyzing expected exceptions can help find better solutions. There are two sources for such analysis: 1) the exceptions occurrence history that records the special events [31] and 2) the history of workflow executions that contains the execution information [30]. The first is useful for studying unexpected exception types and properties (see section 2.1), and their occurrence frequencies. The second is useful for finding and generating the suitable handling methods, at least based upon the prior experiences.

For any of the three reasons for analyzing exceptions (section 2.3), there are four things to consider: data source, the features we want to verify for its properness and effectiveness, exception analysis method and how to use the analysis result.

The frequency of an exception occurrence has been used to estimate the necessity to evolve the workflow definition to manage the exception. Quaglini et al. was able to extract the occurrences of an exception by workflow mining on the workflow execution logs [36]. They used the history information to create the predictability of exceptions. If the frequency is above a certain level, the exception might be considered not a special event, but instead an alternative branch in the workflow definition. An expected, but possibly a rare exception will still be handled as a regular expected exception [31].

The activities necessary to handle an exception can be mined from the history of workflow executions. A workflow management system, which allows ad-hoc adjustment of the individual workflow instances, can have a documented workflow history that may include reusable solutions to react to an unexpected exception (based upon the previous experiences). In case of a new exception, the system can mine exception handling data from historical workflow executions. Hwang and Tang have designed a set of mining algorithms for this purpose [25]. They use a large set of exceptions and runtime data to build up a set of trees for past exceptions according to different attributes and find the possible handling according to the distance between the new exception and the past experience in the trees. Luo et al. described this case-based reasoning methodology in their paper [30] and they provide more details in [29]. They follow four steps as Aamodt et. al discussed in [1] to explore the case repository:

- Retrieve the most similar cases (i.e., events) from database;
- Reuse the knowledge (such as what medicines were given) for handling those cases;
- Adapt the solution of the similar cases to fit the current one;
- Add the verified solution into the case database.

The current data mining technologies are mature and applicable to workflow exception history. A prototype in the METEOR project [4][30] and a branch implementation in a commercial applications like TeamLog [15] showed that the mining algorithm in workflow history works well. However, since the medical domain is highly safety-critical, to use such technologies in practice still needs more efforts (some of them might not be technical, but legal). For example, a physician confirmation for using the exception handling will be necessary.

Data mining can also help to identify the root cause of the exceptions. Thus, the root causes of the exceptions could be removed if their handling can be modified to become methods to prevent the exceptions. Especially, if data dependencies between exceptions and workflow activities/data are explicitly established and modeled, the related activities and data changes can be easily identified for the exceptions. As a result, an extended analysis of the related activities or data changes in healthcare environment or within the computerized system, can lead to a full removal or reduction of the number of caused exceptions. The removal of the root cause is described by Tucker and Edmondson in [41] for improving medical applications.

Strengths and Limitations:

- *Make use of existing data mining methods.* The data mining methods are a research field of data-mining, rather than a part of the workflow research area. How to produce the necessary data for mining is more an implementation issue.
- *The steady improvement of the workflow definition as a result of data-mining is technologically feasible.*

4. Further Research Topics

Based upon our understanding of the current state-of-the-art as described in section 3, we propose three open questions as the major challenges of research on exceptions for medical workflows:

1) *Exception verification;*

2) Exception visualization; and

3) Dynamic exception handling.

Solving those issues effectively would help to make a workflow management system more usable in the medical domain (though likely being used in other areas (e.g., building management systems) as well).

4.1 Exception Verification

Exception verification is the verification of exception and handling definitions against constraints that include syntax, executability, completeness and semantics. There are two types of software verification: static and dynamic. Our discussion here focuses on static verification; by automatically analyzing the software code, derive information about the software execution behavior to establish some correctness criteria.

Developing large scale, consistent workflow systems that have sophisticated exception definitions and handlings is difficult. First, exception management is more likely to be developed with the participation of medical professionals who have limited knowledge of workflow design. Also, exceptions are often built up incrementally by different users, thus there may be conflicts or duplications among different workflows that have exception handlings. Furthermore, certain exceptions can happen anytime during the execution of multiple medical workflows. When the handling of one such exception may be proper for most workflow instances, it may be inappropriate for some specific ones. Thus, identifying errors in exception management are difficult. Since the errors during runtime for medical applications can potentially harm the patients, it is necessary to eliminate as many errors as possible by exception verification. The key features we would need to verify include:

- **Completeness**: Expected exceptions should map to specific handlers as much as possible. A detectable,

unexpected exception may map to a special handler that either enables users to determine handling actions on the fly or is intelligent enough to handle the exception by itself. Any exception which is not otherwise handled should at least be treated by a dynamic human interaction at the root of the exception propagation hierarchy.

- **Conflict-free**: If two exceptions may be raised at the same time, their handling should not conflict. For example, the handling of exception *A* prescribes medicine X while the handling of exception *B* prescribes medicine Y. If *A* and *B* may be raised at the same time for one patient, medicine X and Y should not conflict, i.e. they can be taken concurrently.

- **Compatibility**: The exception handling should not break the constraints of workflow definition (e.g., whenever the workflow is still expected to be continued after the exception). For example, the handling for exception "XRayNotAvailableException" is to use activity "CT chest film", which requires two hours. This would require that the activity after "x-ray exam" be started two hours later to allow for the exception handling time of 'CT chest film'.

- **Correctness**: If an exception handling uses context variables that represent the data only available from where the exception is raised, we need to verify if these variables are accessible and have values for all workflow instances that may raise the exception.

As we discussed in section 3.3, there is little research for verifying exceptions in workflow systems (i.e., to derive the exception handlings' behaviors at execution time and check if they conform to certain correctness criteria). We have also studied exception verification in programming languages. Research in verification of exceptions is limited to completeness ("reliability" in [19]) and the estimate of response time for concurrent exceptions [44]. In computer programs, exception handling is largely to report errors and their sources. It is rare to use exception handling to "repair" or "compensate" an abnormally behaved process or to extend it to deal with the exception conditions, which is however often desired in the medical domain.

Medical knowledge bases should be useful for exception verifications since they can provide rules to be the basis for the correctness criteria to be verified. There are other aspects we want to check, such as whether the exception handling actions fit with the medical guidelines or not (where the guidelines can be a knowledge base). But such verifications probably cannot be performed completely automatically and will need qualified medical staff to accomplish them.

4.2 Exception Visualization

Exception visualization is to represent exception definitions and handlings in a highly visualized form so that users of different kinds can understand them.

The exception management definition alone can be complex with raising conditions and handling actions. In more advanced scenarios dozens of exceptions are possible within one diagram. Simply to add more exceptions to medical workflow diagrams can make the workflows difficult to understand even for IT professionals because exceptions can arise at any time. Furthermore, medical staffs cannot properly handle the exceptions and correctly maintain the exception management definitions, unless they have good understanding of the exception raising condition, origins, handling steps and workflow contexts (e.g., cause-analysis). Thus exception visualization is an important issue for improving the usability of exception managements in medical workflows.

We summarize the key features for visualizing exceptions as:

1. The visualization can provide a view to show the context and other factors for medical staff to determine what exception handling actions should be taken.

2. Support different views of the exception definition. Sample views include workflow-oriented view (given a specific workflow, what exceptions could potentially be raised and handled), exception-oriented view (given specific exceptions raised, what workflows will be impacted), data relation view (given certain data changes, what exceptions could be raised). Such visualizations will aid end users to better understand the exceptions and their relations with other workflows.

3. To provide views to different levels so users can "zoom-in" and "zoom-out" to view the different levels of abstractions.

We discussed the current research status on exception visualization in section 3.1 and have not found any workflow systems that address all of these three points. Visualization techniques for workflow instances [2] may be helpful for resolving the first point.

4.3 Dynamic Exception Handling

Dynamic exception handling is to create a plausible handler upon the raise of an exception and use the handler to deal with the exception. The handler might not be explicitly defined from the beginning to end, but rather is incrementally determined and scheduled as it is being executed.

Medical staffs must response to a wide variety of medical exceptions. Some of them are expected while some are much less expected. Some of them are clinical-related while some of them are hospital administrative or operational-related. They deal with such exceptions in two ways: 1) handling the raised exceptions with skills which are acquired through training and 2) reducing the possibility that the exceptions will occur for the future. For the first way, medical staffs use their past experience, information about the available resources (e.g., medical devices, other physicians), prediction of outcomes to find a

plausible way for handling exceptions under the current circumstances. However, in the longer term, they will use the second way by removing root cause of the exceptions (such as shortage of lab personnel) and/or by improving their processes.

We expect that medical workflow systems should support the above described activities. We understand some of those dynamic exception handlings are probably never supportable by computer software (e.g., some surgical procedures on the patients). However, we also believe that, as the healthcare environments become more and more computerized (wirelessly networked sensors, integrated imaging systems, a wider use of electronic patient medical records), it should be possible to generate some exception handlers for certain exceptions or at least a part of those handlers. For example, the handling actions can call emergency services, order medications, book operating rooms, make lab orders, set medical device configurations/parameters, retrieve patient data, track and report the patients' or medical staffs' locations, etc.. If the workflow management systems can intelligently generate, properly sequence, and execute the handling actions, this would certainly reduce the response time, Further, dynamic handling should achieve the key features as:

- **To support dynamic exception handling**: handling actions could be dynamically customized based upon the information about medical staffs' workload/schedule, resources, past experiences in handling similar exceptions for the patient's best interest. The dynamic exception handling can more likely be applied first to hospital administrative or operational workflows since they require less intelligence and are more likely already computerized.

- **To support the evolution of exception handling**: An embedded exception analysis mechanism should provide support for evolving exception management defined in workflow management systems. The analysis results can be used to remove root cause, perform workflow evolution and/or exception evolution (e.g. frequently detected unexpected exceptions will be defined as expected exceptions).

4.4 An Application Scenario

Our purpose of introducing this scenario is to illustrate how exception techniques, particularly the proposed ones in this paper, can help medical workflow development. The background of our scenario is that a workflow management system is used to support the acute abdominal pain diagnosis workflow in an emergency department (ED).

Patient B comes to the emergency department due to abdominal pain. The patient check-in workflow starts an acute abdominal pain diagnosis workflow instance for patient B, which will automatically retrieves the relevant patient medical records for an attending physician. The physician checks B's records, performs a thorough physical exam and by using the workflow, electronically orders a series of lab exams and imaging exams for the

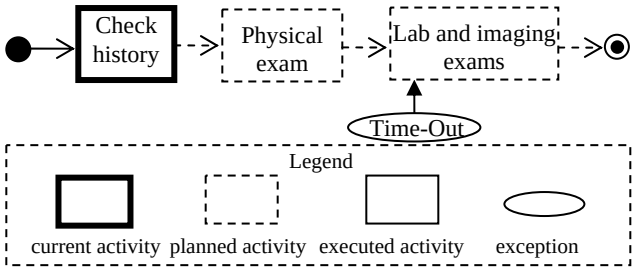


Figure 6 Acute Abdominal Pain Diagnosis Workflow

patient. B finishes the lab exams but waits for 30 minutes and still did not have the CT films taken. A "CT time out" exception is triggered by the radiology information system that is integrated with the workflow system.

Since the workflow system cannot find a predefined handler and there was no automatic solution available, the "ad-hoc handling" mechanism is used. Thus, a pop-up window informs a nurse about the current exception and asks for solution. The nurse may check with the patient status again, and determine that the patient cannot move and a portable CT device is required. But, according to the equipment tracking software, all portable CT devices are in use.

The nurse checks with a physician to select "use alternative resource" and assigns use of US (Ultrasound) instead of CT for this activity. The physician explains to the patient about the change by using a visual form of exception handling and calls the business/insurance people to ensure the insurance covers the cost. Finally the diagnosis workflow ends and B is admitted to in-patient department.

The hospital has limited number of CT devices so the exception of "CT device not available" occurs frequently during this diagnosis workflow. After studying the exception log and discussing with other physicians, the director of the ED decides to add use US instead of CT when CT is not available as a predefined exception handler into the workflow.

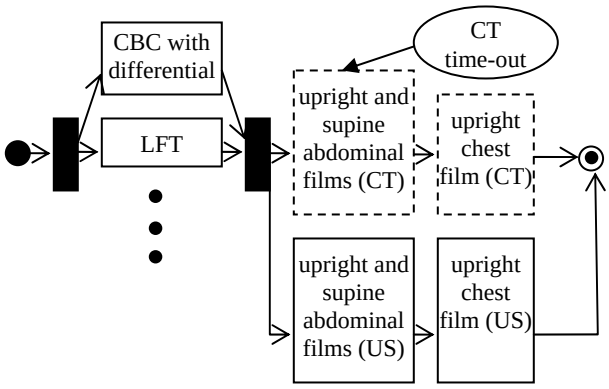


Figure 7 Sub-flow of Lab and Imaging Exams

The director opens a GUI interface to input the exception properties (type, condition, handling actions, etc). The

workflow system automatically verifies the compatibility of this new exception with the rest of the exceptions and the workflow process. Checks are made to see if the handling of the exception conflicts with the knowledge base, the frequency of such exceptions and how it was handled in the past. Since there is no problem found, this exception was successfully added.

Following, we discuss the exception techniques used in this sample scenario with respect to three aspects: exception representation, exception handling and exception analysis.

Exception representation needs to be easy to understand for a variety of users (e.g. nurse, business administration people, doctors, and patients) and maintained by a non-IT staff. From this scenario, we can see that a physician needs to know what exception happened and what handling will be taken for the exception. Manual activities (e.g., patient assessment) need to be intertwined with automated activities. We could see the need for three types of visualization requirements from the above sample scenario:

- When an exception occurs and the workflow instance starts the exception handling process, physicians or hospital administrative staff may want to see the direct cause, the handling actions and the workflow context. This helps them to understand why certain handling actions have been suggested through the alerts or work items. This helps them decide if they should perform the suggested tasks. A re-scheduling mechanism will be needed to determine if the required resources will be available for the timely care.

- A physician is dealing with a newly detected unexpected exception. The medical workflow management system needs to present the physician a view with sufficient information to define the set of handling actions within minutes. For example, he can add a CT check as a replacement when the x-ray device reports that it is busy.

- A physician explains the treatment workflow to a patient and shows the patient the possible exceptions and handling actions during the workflow. Such education to patients is very useful particularly for the therapy procedures. Similarly, it can be very useful to train the medical staff.

Two types of handling mechanisms are used in this scenario: **pre-defined handling** and **ad-hoc handling.** Predefined handling supports the physician setting the handling actions before the exception arises. Ad-hoc handling allows the physician to pick the handling actions on the fly. However, if **dynamic exception handling** is used, when "CT time-out" arises, the system can automatically find the possible alternative choice is US according to the knowledge base and past experience and, and inform nurse and physician of this suggestion.

The exception verification can be used to assure the correctness of the modified workflows after the physician selects or defines exception handling on the fly (such as in the "CT time out" scenario, using US instead of CT) and adds new exception handling afterwards. It is important to verify the exceptions for the 4C rules (completeness, conflict-free, compatibility and correctness). If the director adds the exception of "CT time-out" into the workflow but forgets to add the corresponding handling, a completeness warning will be issued. Then the newly added exception handling is checked against other possible exceptions raised in the same workflow to see if there is any conflict. The compatibility check insures this new exception handling is compatible with the original workflow. If this workflow was originally to end within 2 hours but the new added exception handling, using ultrasound, may end within 2.5 hours, a compatibility warning is given. The correctness check verifies the workflow data used in the exception handling.

5. CONCLUSION

Though managing exception situations is well recognized as daily occurrences in the highly dynamic healthcare environment, computer software systems, particularly the workflow management systems have not been well applied to support such activities. Many workflow systems provide exception management mechanisms of varied capabilities; however, they have not widely been used to support exceptional situation handling. One major reason, we believe, is that at this stage, the healthcare environment has not been fully computerized and many exception handlings mandate human participation, skills and reasoning. On the other hand, we also believe that as the healthcare environment will be far more computerized in the next few years, this problem will decrease. With computerized healthcare environments, more exception situations or more parts of them would be detectable and dynamically handled by workflow systems that are integrated with computerized systems such as financial systems, drug ordering systems, electronic patient record systems, medical devices, disease treatment devices, and other healthcare information systems.

We started research on exception management for medical workflows about two year ago. We believe that, in a short term, representing and verifying exception management in medical workflows should help improve the execution and quality of the medical procedures. In a longer term, dynamic exception handling and workflow re-scheduling should improve the efficiency of the healthcare providers to respond to unexpected exceptions.

6. ACKNOWLEDGEMENTS

This work was sponsored by and carried out at the Siemens Corporate Research. We are very grateful to the insightful comments provided by Beatrice Hwong, Arnold Rudorfer, Gilberto Matos, the colleagues from Siemens Medical and Hui Cao from Biomedical Informatics Department of Columbia University. Also thanks to Juergen Kazmeier for financial support for this research.

7. REFERENCES

[1] Aamodt, A. and Plaza, E. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, 7, 1, (1994), 39-59.

[2] Aigner, W. and Miksch, S. Supporting Protocol-Based Care in Medicine via Multiple Coordinated Views. In *Proceedings International Conference on Coordinated and Multiple Views in Exploratory Visualization*, (2004), 118-129.

[3] Anderson, T., and Lee, P. A. *Fault Tolerance Principles and Practice*, Prentice Hall International, 1981.

[4] Anyanwu, K., Sheth, A., Cardoso, J., Miller, J., and Kochut, K. Healthcare Enterprise Process Development and Integration. Technical report, University of Georgia, 2002.

[5] Ardissono, L., Leva, A. Di, Petrone, G., Segnan, M., and Sonnessa, M. Adaptive Medical Workflow Management for a Context-Dependent Home Healthcare Assistance Service. In *Electronic Notes in Theoretical Computer Science*, Elsevier, (2005).

[6] Arkin, A. Business Process Modeling Language. (2002).

[7] Buhr, P. A. and Mok, W. Y. R. Advanced Exception Handling Mechanisms. *IEEE Transactions on Software Engineering*, 26, 9, (2000), 820-836.

[8] Casati, F., *Models, Semantics, and Formal Methods for the design of Workflows and their Exceptions*. PhD thesis, Politecnico Di Milano, (1998).

[9] Casati, F., Ceri, S., Paraboschi, S., and Pozzi, G. Specification and Implementation of Exceptions in Workflow Management Systems. *ACM Transactions on Database Systems*, 24, 3, (1999), 405-451.

[10] Chiu, D. K. W., Karlapalem, K., and Li, Q. Exception Handling with Workflow Evolution in ADOME-WFMS: a Taxonomy and Resolution Techniques. In *CSCW Workshop: Towards Adaptive Workflow Systems*, (Nov 14-18, 1998), Seattle, Washington.

[11] Cook, K. Evaluating acute abdominal pain in adults http://www.jaapa.com/issues/j20050301/articles/belly0305.htm

[12] Dadam, P. and Reichert, M. Towards a new dimension in clinical information processing. *Stud Health Technol Inform*, 77, (2000), 295-301.

[13] Dazzi, L. and Stefanelli, M. A patient workflow management system built on guidelines, In *Proc. of AMIA 97*, (1997), 146-150.

[14] Deiters, W., Goesmann, T., Just-Hahn K., Lefeler, T. and Rolles, R. Support for exception handling through workflow management systems. *CSCW Workshop: Towards Adaptive Workflow Systems*, (1998).

[15] Dustdar, S., Hoffmann, T., and Aalst, W.v.d. Mining of ad-hoc Business Processes with TeamLog. *Data and Knowledge Engineering*, Elsevier, (Sep 2005).

[16] Edelweiss, N. and Nicolao, M. Workflow Modeling: Exception and Failure Handling Representation. In *SCCC '98: Proceedings of the XVIII International Conference of the Chilean Computer Science Society*, (1998), 58.

[17] Eder, J. and Liebhert, W. Contributions to Exception Handling in Workflow Management. In *Proceedings EDBT Workshop on Workflow Management Systems*, (1998), 3-10.

[18] Eder, J. and Liebhart, W. The Workflow Activity Model WAMO. In *Proceedings of 3rd International Conference on Cooperative Information Systems*, (1995).

[19] Garcia, A. F., Rubira, C. M. F., Romanovsky, A., and Xu, J. A Comparative Study of Exception Handling Mechanisms for Building Dependable Object-Oriented Software. *Journal of Systems and Software, Elsevier*, 59, 2, (Nov 2001), 197-222.

[20] Georgakopoulos, D., Hornick, M., and Sheth, A. An overview of workflow management: from process modeling to workflow automation infrastructure. *Distributed Parallel Databases*, 3, 2, (1995), 119-153.

[21] Greiner, U., Ramsch, J., Heller, B., Löffler, M., Müller, R., Rahm, E. Adaptive Guideline-based Treatment Workflows with AdaptFlow. In *Proc. of Symposium on Computerized Guidelines and Protocols*, (2004), 113-117.

[22] Grigori, D. Improving Business Process Quality through Exception Understanding, Prediction, and Prevention. In *Proceedings of the 27th International Conference on Very Large Data Bases*, (2001), 159 - 168.

[23] Hagen, C. and Alonso, G. Exception Handling in Workflow Management Systems. *IEEE Transactions on Software Engineering*, 26, (2000), 943-958.

[24] Hagen, C. and Alonso, G. Flexible Exception Handling in the OPERA Process Support System. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS)*, (1998), 525-533.

[25] Hwang, S., and Tang, J. Consulting past exceptions to facilitate workflow exception handling. *Decision Support Systems*, 37, 1, (2004), 49-69.

[26] Klein, M., Dellarocas, C. A Knowledge-based Approach to Handling Exceptions in Workflow Systems. *Computer Supported Cooperative Work*, 9, 3-4, (2000).

[27] Kobayashi, M., Fussell, S. R., Xiao, Y., Seagull, F. J. Work coordination, workflow, and workarounds in a medical context. Conference on Human Factors in Computing Systems, (2005).

[28] Laprie, J. C. Ed. Dependability: Basic concepts and Terminology. Volume 5 of Dependable Computing and Fault-Tolerant Systems, Springer-Verlog, (1992).

[29] Luo, Z., Sheth, A., Kochut, K., and Arpinar, B. Exception Handling for Conflict Resolution in Cross-Organizational Workflows. *Distributed and Parallel Databases*, 13, (2003), 271-306.

[30] Luo, z., Sheth, A., Kochut, K., and Miller, J. Exception Handling in Workflow Systems. *Applied Intelligence*, 13, 2, (2000), 125-147.

[31] Medeiros, A.d., Aalst, W.v.d. and Weijters, A. Workflow Mining: Current Status and Future Directions. In *On the Move to Meaningful Internet Systems*, (2003), Z. Tari R. Meersman and D.C. Schmidt, ed.

[32] Ogasawara, T., Komatsu, H., and Nakatani, T. A study of exception handling and its dynamic optimization in Java. In *Proceedings of the 16th ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages, and Applications* (OOPSLA '01), (2001), 83-95.

[33] Peleg, M., Boxwala, A. A., Ogunyemi, O., Zeng, Q., Tu, S., Lacson, R., Bernstam, E., Ash, N., Mork, P., Ohno-Machado, L., Shortliffe, E. H., Greenes, R. A. GLIF3: the evolution of a guideline representation format. In *Proc AMIA Symp.*, (2000), 645-653.

[34] Poulymenopoulou, M. and Vassilacopoulos, G. A Web-based Workflow System for Emergency Healthcare. *Medical Informatics Europ*, (2002).

[35] Quaglini, S., Caffi, E., Cavallini, A., Micieli, G., and Stefanelli, M. Simulation of a Stroke Unit Careflow, Medinfo 2001.

[36] Quaglini, S., Stefanelli, M., Lanzola, G., Caporusso, V., and Panzarasa, S. Flexible guideline-based patient careflow systems. *Artificial Intelligence in Medicine*, 22, 1, (2001), 65-80.

[37] Song, X., Hwong, B., Matos, G., Rudorfer, A., Nelson, C., Han, M., Girenkov, A., Understanding requirements for computer-aided healthcare workflows: experience and challenges. In Proceedings of the 28th international conference on Software engineering, Shanghai, China, pages 930-934, 2006.

[38] Stiphout, R. van, Meijler, T. D., Aerts, A., Hammer, D., and Comte, R. le. TREX: Workflow TRansactions by Means of Exceptions. In *Proceedings of WFMS'98 EDBT Workshop on Workflow Management Systems*, (1998).

[39] Sutton, D. R., and Fox, J. The Syntax and Semantics of the PROforma Guideline Modeling Language. *J Am Med Inform Assoc.*, (Sep-Oct, 2003), 10, 5, 433-476.

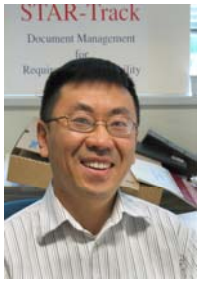
[40] Tu, S. W. and Musen M. A. A flexible approach to guideline modeling. In *Proc AMIA Symp.*, (1999), 420-424.

[41] Tucker, A. L., and Edmondson, A. Managing Routine Exceptions: A Model of Nurse Problem Solving Behavior. *Advances in Health Care Management*, 3, (2002), 87-113.

[42] WfMC. Workflow Process Definition Interface - XML Process Definition Language. Workflow Management Coalition Workflow Standard, (2002).

[43] Widom, J. and Ceri, S., Eds. Active Database Systems. Morgan Kaufmann Publishers Inc., San Francisco, CA. (1996).

[44] Xu, J., Romanovsky, A., and Randell, B. Coordinated Exception Handling in Distributed Object Systems: From Model to System Implementation. In *Proceedings of the 18th international Conference on Distributed Computing Systems* (ICDCS). IEEE Computer Society, Washington, DC, 12, (1998)



Xiping Song received B.S. degree in computer science in 1982 from Beijing Polytechnic University, Beijing, China, M.S. from University of Colorado at Boulder, and Ph.D. degrees in computer science in 1992 from University of California at Irvine.

He is a senior research scientist at Software & Engineering Department of Siemens Corporate Research. Dr. Xiping Song has over 15 year industrial application and research experiences in the software process, software design, requirement engineering, healthcare information systems and web-based software applications. He also published numerous papers and hold patents in the above areas.

Minmin Han received B.S. degree in computer science in 1999 from Tongji University, China, M.S. degree in computer science in 2002 from Shanghai Jiao Tong University, China and Ph.D. degrees in computer science in 2006 from Lehigh University, USA.

Currently she works for Amazon.com. She was a research assistant and teaching assistant in Lehigh University and an intern in Siemens Corporate Research. She has published papers on software modeling, web engineering, and medical information systems.

She is the recipient of the Best Paper Award from the 5th International Conference on Web Engineering (ICWE 2005).

Thomas Thiery is a graduate student in computer science and will receive his diploma within 2007 from the Technische Universität München, Germany. He formerly received a vocational title as technical assistant for computer science in 1999 from the Siemens Technik Akademie, Munich, Germany. He is also a consultant and specialist for web based workflows.