Information Technology and Management 1 (2000) 155–169

A knowledge-based approach for handling exceptions in business processes

Chrysanthos Dellarocas and Mark Klein

Center for Coordination Science, Sloan School of Management, Massachusetts Institute of Technology, Room E53-315, Cambridge, MA 02139, USA
E-mail: {dell;m_klein}@mit.edu

This paper describes a novel knowledge-based methodology and toolset for helping business process designers and participants better manage *exceptions* (unexpected deviations from an ideal sequence of events caused by design errors, resource failures, requirement changes, etc.) that can occur during the enactment of a process. This approach is based on an on-line repository exploiting a generic and reusable body of knowledge, which describes what kinds of exceptions can occur in collaborative work processes, how these exceptions can be detected, and how they can be resolved. This work builds upon previous efforts from the MIT Process Handbook project and from research on conflict management in collaborative design.

Keywords: business process modeling, business process reengineering, exception handling, knowledge-based systems, operational risk management

1. Introduction

Business process models typically describe the "normal" flow of events in an ideal world. For example, the model of a product development process typically includes a "design product" activity, followed by a "build product" activity, which, in turn, is followed by a "deliver product" activity. Reality, however, tends to be more complicated. During the enactment of a business process a lot of *exceptions*, that is, deviations from the ideal sequence of events, might occur. For example, product design might prove to be inconsistent with the capabilities of the manufacturing plant. Manufacturing stations might break down in the middle of jobs. Delivery trucks might go on strike. To assure that a process is still able to fulfill its organizational goals, process participants must be able to detect, diagnose and successfully resolve such exceptional conditions as they occur.

Traditionally, managers have been relying on their experience and understanding of a process in order to handle deviations from the expected flow of events. However, the rising complexity of modern business processes and the accelerating pace with which these processes evolve and change has made the reliance on individual managers' experience and intuition an increasingly less satisfactory way to deal with exceptions. There is an increasing need for *systematic* business process *operational*

risk management methodologies. Such methodologies will assist business process designers to anticipate potential exceptions and instrument their processes so that exceptions can either be avoided or be detected on time. Furthermore, when exception manifestations occur during process enactment, the same methodologies will assist in selecting and instantiating the best way of resolving them.

Business process modeling has been used successfully in order to increase understanding, facilitate analysis and enhance communication among the various stakeholders involved in the design and enactment of an "ideal" business process. Our position is that analogous model-based tools can form the basis of systematic methodologies for operational risk management.

The standard approach of incorporating exception handling in process models has been to try to anticipate beforehand all possible exceptional conditions that might arise and augment an "ideal" process model with additional conditional elements that represent exception handling actions. This approach, however, is problematic for a number of reasons. First, it results in cluttered, overly complex, models, which hinder instead of enhancing understanding and communication. Second, the anticipation of possible failure modes once again relies on the experience and intuition of the model designers. Third, the approach cannot help with exceptions that have not been explicitly hard-coded into the model.

This paper describes a knowledge-based approach for handling exceptions in business processes. Rather than requiring process designers to anticipate all possible exceptions up front and incorporate them into their models, this approach is based on a set of novel computerized process analysis tools, which assist designers to analyze "ideal" process models, *systematically* anticipate possible exceptions and suggest ways in which the "ideal" process can be instrumented in order to detect or even to avoid them. When exception manifestations occur, the same tools can be used to diagnose their underlying causes, and suggest specific interventions for resolving them. The approach is based on an extensible knowledge base of generic strategies for avoiding, detecting, diagnosing and resolving exceptions.

The remainder of the paper is structured as follows: section 2 provides an overview of the proposed approach. Section 3 describes how the approach has been successfully applied to analyze operational risks of the Barings Bank trading processes. Section 4 discusses related work. Finally, section 5 reports on the current status of this work and presents some directions for future work.

2. A knowledge-based approach to exception handling

2.1. Anticipating and preparing for exceptions

The first step in our approach assists process designers to anticipate, for a given "ideal" process model, the ways that the process may fail and then instrument the process so that these failures can be detected or avoided. The principal idea here is to compare a process model against a taxonomy of elementary process elements

Table 1
A subset of exception causes.

Exceptions related to goals and assumptions	Goals contain conflicts or inconsistencies Unanticipated requirement changes violate assumptions
Exceptions related to activities	Wrong process selected for stated goals Process contains design flaws Process contains intrinsic possibilities of conflicts, deadlock, etc.
Exceptions related to resources	Wrong resource assigned to task Resource unavailable Resource fails in the middle of task

annotated with possible failure modes. Our idea is motivated by the observation that the causes of most process failures have a straightforward association with one of the four principal elements of business process models: activities, resources, underlying goals and assumptions. Table 1 lists some examples.

A process element taxonomy can be defined as a hierarchy of process element templates, with very generic elements at the top and increasingly *specialized* elements below. For example, figure 1 depicts a small activity taxonomy. Each activity can have attributes, e.g., that define the challenges for which it is well-suited. Note that activity *specialization* is different from *decomposition*, which involves breaking an activity down into subactivities. While a subactivity represents a part of a process; a specialization represents a "subtype" or "way of" doing the process [10]. Resource, goal and assumption taxonomies can be defined in a similar manner.

Process element templates are annotated with the ways in which they can fail, i.e., with their characteristic *exception types*. Failure modes for a given process template can be uncovered using failure mode analysis [11]. Each process element in a taxonomy inherits all characteristic failure modes of its parent (generalization) and may contain additional failure modes which are specific to it.

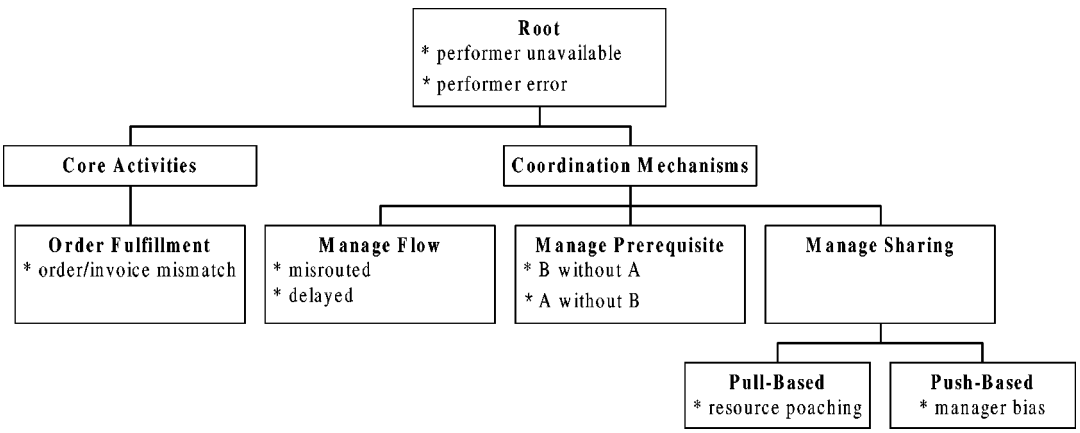

Figure 1. An example of a generic activity taxonomy with failure modes.

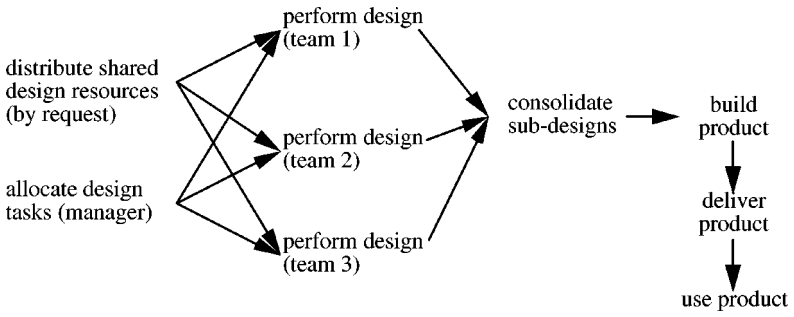

Figure 2. An example "Ideal" process model.

Given an "ideal" process model, to identify failure modes we need only identify the generic process element templates that match each element (activity, resource, goal, assumption) of the model. The potentially applicable exception types will then consist of the union of all failure modes inherited from the matching templates. We can see, for example, that the "distribute shared design resources" activity in figure 2 is a subtype of the generic "pull-based sharing" process template in figure 1, since the resources are "pulled" by their consumers rather than "pushed" (i.e., allocated) by their producers. This template includes among its characteristic failure modes the exception called "poaching", wherein resources go disproportionately to lower priority tasks because agents with lower priority tasks happen to reserve them first. The "deliver product" activity is a specialization of the "manage flow" template, with characteristic exceptions such as "item delayed", "item misrouted" and so on. All activities also inherit the characteristic failure modes from the generalizations of these matching templates, such as "responsible agent is unavailable", and so on.

The process designer can select, from this list of possible exception types, the ones that seem most important in his/her particular context. He/she might know, for example, that the "deliver product" process is already highly robust and that there is no need to augment it with additional exception handling capabilities.

For each exception type of interest, the process designer can then decide how to instrument the process in order to detect these exceptions. While processes can fail in many different ways, such failures have a relatively limited number of different manifestations, including missed deadlines, violations of artifact constraints, exceeding resource limits, and so on. Every exception type includes pointers to *exception detection* process templates in the process taxonomy that specify how to detect the symptoms manifested by that exception type. These templates, once interleaved into the "ideal" process model by the workflow designer, play the role of "sentinels" that check for signs of actual or impending failure. The template for detecting the "resource poaching" exception, for example, operates by comparing the average priority of tasks that quickly receive shared resources against the average priority of all tasks. The "item delayed", "agent unavailable", and "item misrouted" exceptions can all be detected using time-out mechanisms. Similar pointers exist to *exception avoidance* processes, whose purpose is to try to prevent the exceptional condition from occurring at all.

2.2. Diagnosing exceptions

When exceptions actually occur during the enactment of a process, our tools can assist process participants in figuring out how to react. Just as in medical domains, selecting an appropriate intervention requires understanding the underlying cause of the problem, i.e., its *diagnosis*. A key challenge here, however, is that the symptoms revealed by the exception detection processes can suggest a wide variety of possible underlying causes. Many different exceptions (e.g., "agent not available", "item misrouted" etc.) typically manifest themselves, for example, as missed deadlines.

Our approach for diagnosing exception causes is based on heuristic classification [2]. It works by traversing a diagnosis taxonomy. Exception types can be arranged into a taxonomy ranging from highly general failure modes at the top to more specific ones at the bottom; every exception type includes a set of defining characteristics that need to be true in order to make that diagnosis potentially applicable to the current situation (figure 3).

When an exception is detected, the responsible process participant traverses the exception type taxonomy top-down like a decision tree, starting from the diagnoses implied by the manifest symptoms and iteratively refining the specificity of the diagnoses by eliminating exception types whose defining characteristics are not satisfied. Distinguishing among candidate diagnoses will often require that the user get additional information about the current exception and its context, just as medical diagnosis often involves performing additional tests.

Imagine, for example, that we have detected a time-out exception in the "deliver product" step (see figure 2). The diagnoses that can manifest this way include "agent unavailable", "item misrouted", and "item delayed". The defining characteristics of these exceptions are:

- *agent unavailable*: agent responsible for task is unavailable (i.e., sick, on vacation, retired, etc.);
- *item misrouted*: current location and/or destination of item does not match original target destination;
- *item delayed*: item has correct target destination but is behind original schedule.

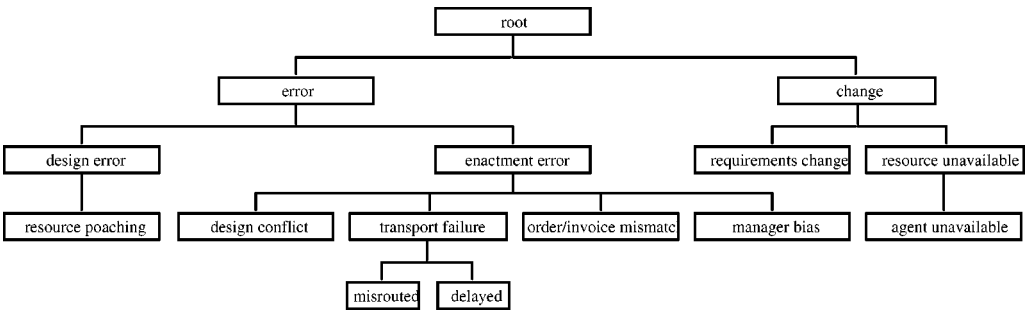

Figure 3. A subset of the exception type taxonomy.

The user then has a specific set of questions that he/she can ask in order to narrow down the exception diagnosis. If the appropriate information is available on-line, then answering such questions and thereby eliminating some diagnoses can potentially be automated.

2.3. Resolving exceptions

Once an exception has been detected and at least tentatively diagnosed, one is ready to define a *prescription* that resolves the exception and returns the process to a viable state. This can be achieved, in our approach, by selecting and instantiating one of the generic exception resolution strategies that are associated with the hypothesized diagnosis. These strategies are processes like any other, are captured in a portion of the process taxonomy, and are annotated with attributes defining the *preconditions* that must be satisfied for that strategy to be applicable. We have accumulated roughly 200 such strategies to date, including for example:

- IF a process fails, THEN try a different process for achieving the same goal;
- IF a highly serial process is operating too slowly to meet an impending deadline, THEN pipeline (i.e., release partial results to allow later tasks to start earlier) or parallelize to increase concurrency;
- IF an agent may be late in producing a time-critical output, THEN see whether the consumer agent will accept a less accurate output in exchange for a quicker respons;
- IF multiple agents are causing wasteful overhead by frequently trading the use of a scarce shared resource, THEN change the resource sharing policy such that each agent gets to use the resource for a longer time;
- IF a new high-performance resource applicable to a time-critical task becomes available, THEN reallocate the task from its current agent to the new agent.

Since an exception can have several possible resolutions, each suitable for different situations, we use a procedure identical to that used in diagnosis to find the right one. Imagine, for example, that we want a resolution for the diagnosis “agent unavailable”. We start at the root of the process resolution taxonomy branch associated with that diagnosis (figure 4).

Three specific strategies are available, with the following preconditions and actions:

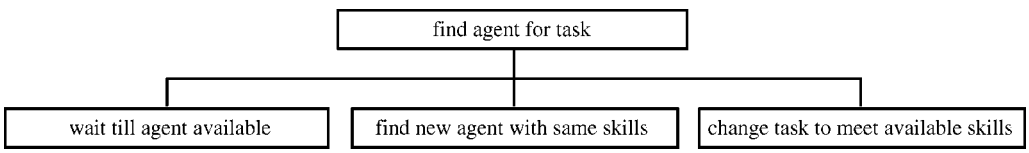

Figure 4. A fragment of the resolution process taxonomy.

- *wait till agent available*: IF the original agent will be available in time to complete the task on the current schedule THEN wait for original agent to start task;
- *find new agent with same skills*: IF another agent with the same skills is available, THEN assign task to that agent;
- *change task to meet available skills*: IF the task can be performed a different way using agents we have currently available THEN modify and re-assign.

The system user can prune suggested strategies based on which preconditions are satisfied, and enact or customize a strategy selected from the remainder. Note that the substantial input may be needed from the user in some cases in order to instantiate a strategy into specific actions.

2.4. Summary

Figure 5 summarizes the knowledge structure which serves as the basis of the approach described in the previous sections. It consists of two cross-referenced taxonomies: a specialization taxonomy of process model entities (activities, resources, goals, assumptions) and a taxonomy of exception types.

During process design time, process models are compared against the process taxonomy in order to identify possible failure modes. Once failure modes are identified, the exception type taxonomy provides links to appropriate detection and avoidance processes. During process enactment time, exception manifestations are compared against the exception type taxonomy in order to identify possible diagnoses. Once plausible diagnoses have been identified, the exception taxonomy provides links to resolution processes.

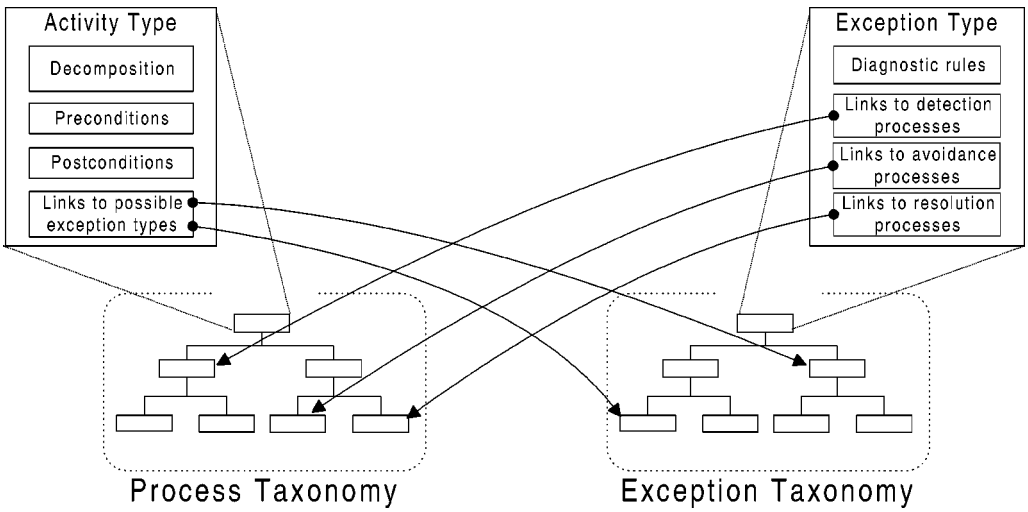

Figure 5. Overview of exception handling knowledge structures.

3. Case study: Barings Bank

The approach described in the previous section can be applied in order to help design robust new processes. It can also be a helpful tool when testing the robustness of existing business processes. This section illustrates how the method has been used in order to systematically expose potential dangers (and suggest possible fixes) in a well-known case of a failed business process.

In February 1995, 233-year old Barings Bank, one of the oldest and most respected investment houses in the United Kingdom, went bankrupt. The entire bank collapsed because of losses of $1.4 billion incurred in a matter of days by a single young trader, Nicholas Leeson. Nicholas Leeson was a futures trader in the Singapore branch of the bank. For a number of reasons, which are still not entirely clear, Leeson began to engage in unauthorized futures trading in the Singapore exchange. Due to inadequate internal controls and other process failures, Leeson was able to maintain his unauthorized and highly risky activity undetected by the bank headquarters in London until the very end.

The collapse of the Barings Bank is one of the most dramatic and talked about recent disasters in financial markets. There exist several detailed accounts and analyses of why and how it happened (for example, [5,12]). From our perspective, the Barings disaster is interesting because it was the result of a series of undetected exceptions in one of the bank's secondary business processes: the futures trading process in Singapore.

In this section, we will demonstrate how the approach described in this paper can be used to systematically point out the gaps in the Barings trading process controls, as well as to suggest ways for closing those gaps.

As described in the previous section, the approach begins with an "ideal" model of the process. Figure 6 depicts a simplified but accurate model of the futures trading process, based on the descriptions contained in [5] and [12]. The model consists of boxes, which describe process activities, and lines, which describe various dependency relationships, that is, constraints that must hold true in order for the process to succeed. The following is a brief description of the process: When a customer requests a futures trade, the trader asks the bank headquarters for an advance of funds in order to cover the customer's margin account[1]. Once the funds have arrived, the trader performs the trade, waits to receive the corresponding security certificate and finally pays the exchange. In an "ideal" world, a trader only performs trades when authorized to do so by customers, correct certificates are always received, and payment for trades exactly match the funds forwarded to the trader by the bank headquarters. These conditions are implied by the "prerequisite" and "exact flow" relationships, which are part of the "ideal" process model.

The first step in our exception handling methodology consists of identifying the possible exceptions that are associated with each element of the "ideal" process model.

[1] To find out more about derivatives trading and the meaning of margin accounts, the interested reader is referred to Zvi Bodie, Alex Kane, Alan J. Marcus, *Investments* (4th Edition), Irwin, 1998 (Part IV).

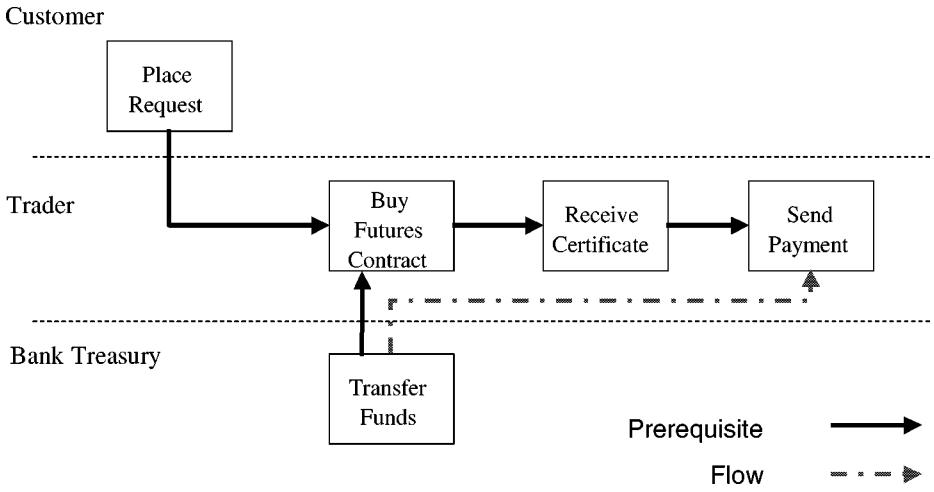

Figure 6. The Barings futures trading process.

For simplicity we will only consider here exceptions associated with dependency relationships in the model.

According to the failure mode taxonomy shown in figure 1, one possible exception of any prerequisite relationship is a prerequisite violation ("B without A"), that is, the possibility of activity B happening without a prior occurrence of activity A. In the context of the Barings trade process such violations would translate into unauthorized trading, unwanted security receipts and unnecessary payment (figure 7).

Likewise, one possible exception of an "exact flow" process is mismatch between the amount produced and the amount consumed. In the context of the Barings process this would translate into a misuse of headquarter funds.

After possible exceptions have been identified, the next step is to use the information stored in the exception type taxonomy (figure 2) in order to find ways for avoiding or detecting the exceptions. It turns out that, because the trading process at Barings involves several independent entities (customer, bank, exchange) and requires some initiative from the part of the trader, there were no practical mechanisms for avoiding the exceptions. There were, however, several mechanisms for detecting them.

For example, *logging* is one (out of several) generic mechanism for detecting prerequisite relationship violations (figure 8). Logging involves recording all occurrences of activities A and B in some reliable storage medium and periodically conducting checks for prerequisite violations. In order for logging to be successful it is, in turn, required that (a) all occurrences of A and B are reliably logged and (b) the log can only be modified by the processes that do the logging.

If we insert a logging process for all dependencies listed in figure 6 we get a model of a properly instrumented trading process (figure 9).

At this point, we can compare the process derived using our approach with the actual Barings described in [5,12]. It can immediately be seen that, although Barings

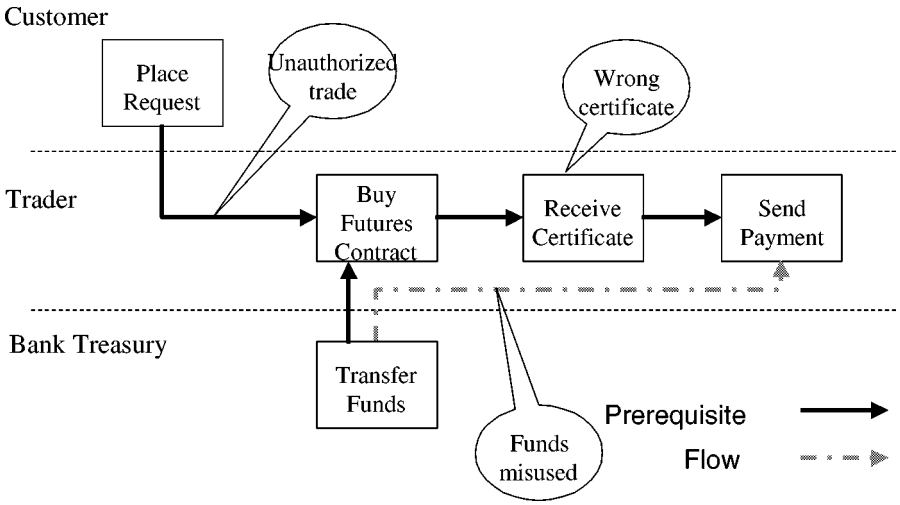

Figure 7. The Barings futures trading process with associated exceptions.

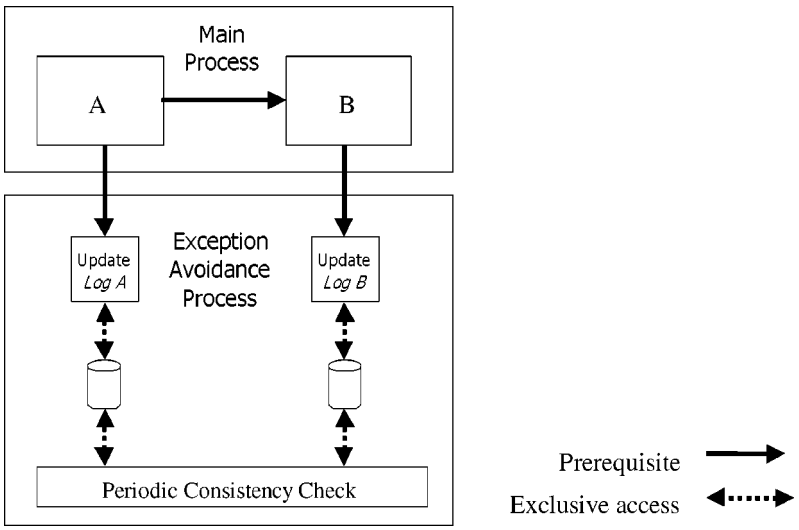

Figure 8. Logging is a generic process for detecting prerequisite violations.

did log some information about trades, it had two crucial gaps relative to the properly instrumented process of figure 9 (see figure 10):

First, it failed to log and compare the amount of funds forwarded by headquarters to the trader to the amounts actually paid by the trader for customer trades (in other words, the log labeled “Funds” in figures 9, 10 was missing from the Barings process). Second, Nick Leeson, in addition to being a trader, was also in charge of the back room operations in the Singapore branch. This gave him the authorization to modify the trades logs (and thus violated requirement (b) above of the logging process).

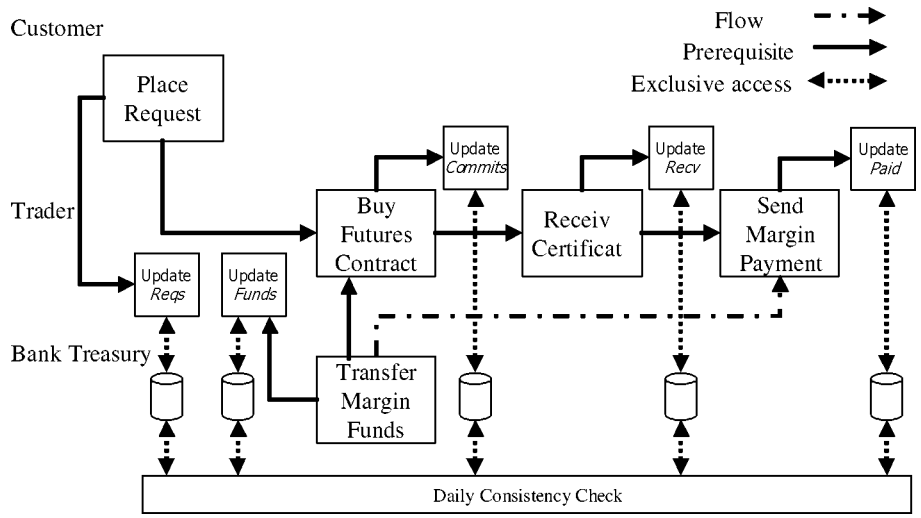

Figure 9. Barings process properly instrumented with logging processes.

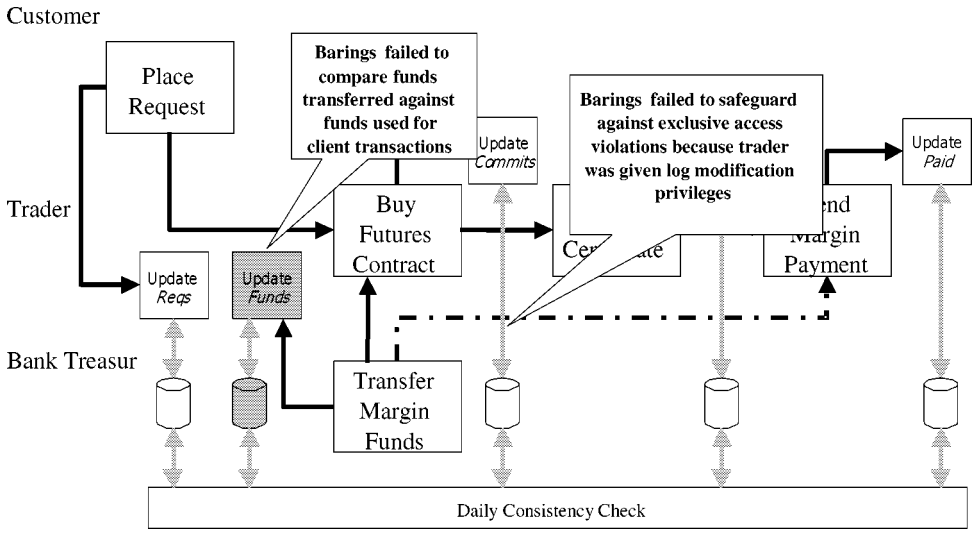

Figure 10. Comparison between ideal and actual Barings process.

Nick Leeson was able to use these two gaps to his advantage as follows: Whenever he received a trade request from a customer, he requested an amount of funds far greater than what was required for the customer trade. He then performed the customer trade, as well as some additional unauthorized trades on his behalf. All of these trades were automatically logged into logs “Commits”, “Received” and “Paid” (see figures 9, 10). Leeson then erased the records of his unauthorized trades from logs “Commits”, “Received” and “Paid”. Therefore, at the end of each day, the log of “Requests” matched perfectly the other three logs. By not checking for discrepancies

between the funds forwarded to Leeson and the total funds recorded at the "Paid" log, headquarters remained unaware of Leeson's activities until it was too late.

It is probably too simplistic to claim that the Barings disaster would have been avoided if the management of Barings had at their disposal knowledge-based exception handling methodologies, such as the ones described in this paper. Nevertheless, this exercise demonstrates that these methodologies and tools can be used in real-life cases to alert management of potential weaknesses and suggest ways for making vital business processes more robust.

4. Related work

The approach described here integrates and extends two long-standing lines of research: one addressing coordination science principles about how to represent and utilize process knowledge, another addressing how artificial intelligence techniques can be applied to detecting and resolving conflicts in collaborative design settings:

One component is a body of work pursued over the past five years by the Process Handbook project at the MIT Center for Coordination Science [3,9,10]. The goal of this project is to produce a repository of process knowledge and associated tools that help people to better redesign organizational processes, learn about organizations, and automatically generate software. The Handbook database continues to grow and currently includes over 4500 models covering a broad range of business processes. A mature Windows-based tool for editing the Handbook database contents, as well as a Web-based tool for read-only access have been developed. A key insight from this work is that a repository of business process templates, structured as a specialization taxonomy, can assist people to design innovative business processes more quickly by allowing them to retrieve, contrast and customize interesting examples, make "distant analogies", and utilize "recombinant" (mix-and-match) design techniques.

The other key component of this work is nearly a decade of development and evaluation of systems for handling multi-agent conflicts in collaborative design [6,7] and collaborative requirements capture [8]. This work resulted in principles and technology for automatically detecting, diagnosing and resolving design conflicts between both human and computational agents, building upon a knowledge base of roughly 300 conflict types and resolution strategies. This technology has been applied successfully in several domains including architectural, local area network and fluid sensor design. A key insight from this work is that design conflicts can be detected and resolved using a knowledge base of generic and highly reusable conflict management strategies, structured using diagnostic principles originally applied to medical expert systems. Our experience to date suggests that this knowledge is relatively easy to acquire and can be applied unchanged to multiple domains.

The work described in this paper integrates and extends these two lines of research in an innovative and, we believe, powerful way. The central insights underlying this

integration are that (1) business process exceptions can be handled by generalizing the diagnostic algorithms and knowledge base underlying design conflict, and (2) the exception handling knowledge base can be captured as a set of *process templates* that can be retrieved, compared and customized using the principles embodied in the Process Handbook. The result of this integration is an approach that allows process designers and participants to better take advantage of insights collected from a wide range of experts and domains when trying to determine what exceptions can occur in their process, as well as how such exceptions can be detected, diagnosed and resolved.

5. Current status and future work

To date, we have captured over 4500 generic process templates, 100 exception types and 200 exception resolution strategies and have constructed a cross-referenced knowledge base with this information on top of the Process Handbook tools (figure 11; see [10] for a description of the Process Handbook tools).

This paper has emphasized the use of our exception handling knowledge base as a decision support tool for humans. Our ongoing work is also focused on connecting our technology with automated process enactment systems, such as workflow controllers and software agent systems. It is widely recognized that state-of-the art workflow technology provides very rudimentary support for exception handling [1,4]. The result of our work will be a prototype implementation of a domain-independent *exception handling engine*, which oversees the enactment of a workflow script, monitors for exceptions and decides (automatically for the most part) how to intervene in order to resolve them. Given an "ideal" workflow script, the engine first uses the exception handling knowledge base in order to anticipate potential exceptions and augment the system with additional actions that play the role of *software sentinels*. During enactment time, these sentinels automatically trigger the diagnostic services of the engine when they detect symptoms of exceptional conditions. The diagnostic services traverse the exception type taxonomy, select (possibly with human assistance) a diagnosis and then select and instantiate a resolution plan. The resolution plan is eventually translated into a set of workflow modification operations (e.g., add tool, remove tool, modify connection, etc.), which are dynamically applied to the executing workflow.

For further information about our work, please see the Adaptive Systems and Evolutionary Software web site at http://ccs.mit.edu/ases/. For further information on the Process Handbook, see http://ccs.mit.edu/.

Acknowledgment

The authors would like to thank Rafael Yahalom for helping them understand the Barings bank processes. The authors gratefully acknowledge the support of the DARPA CoABS Program (contract F30602-98-2-0099) while preparing this paper.

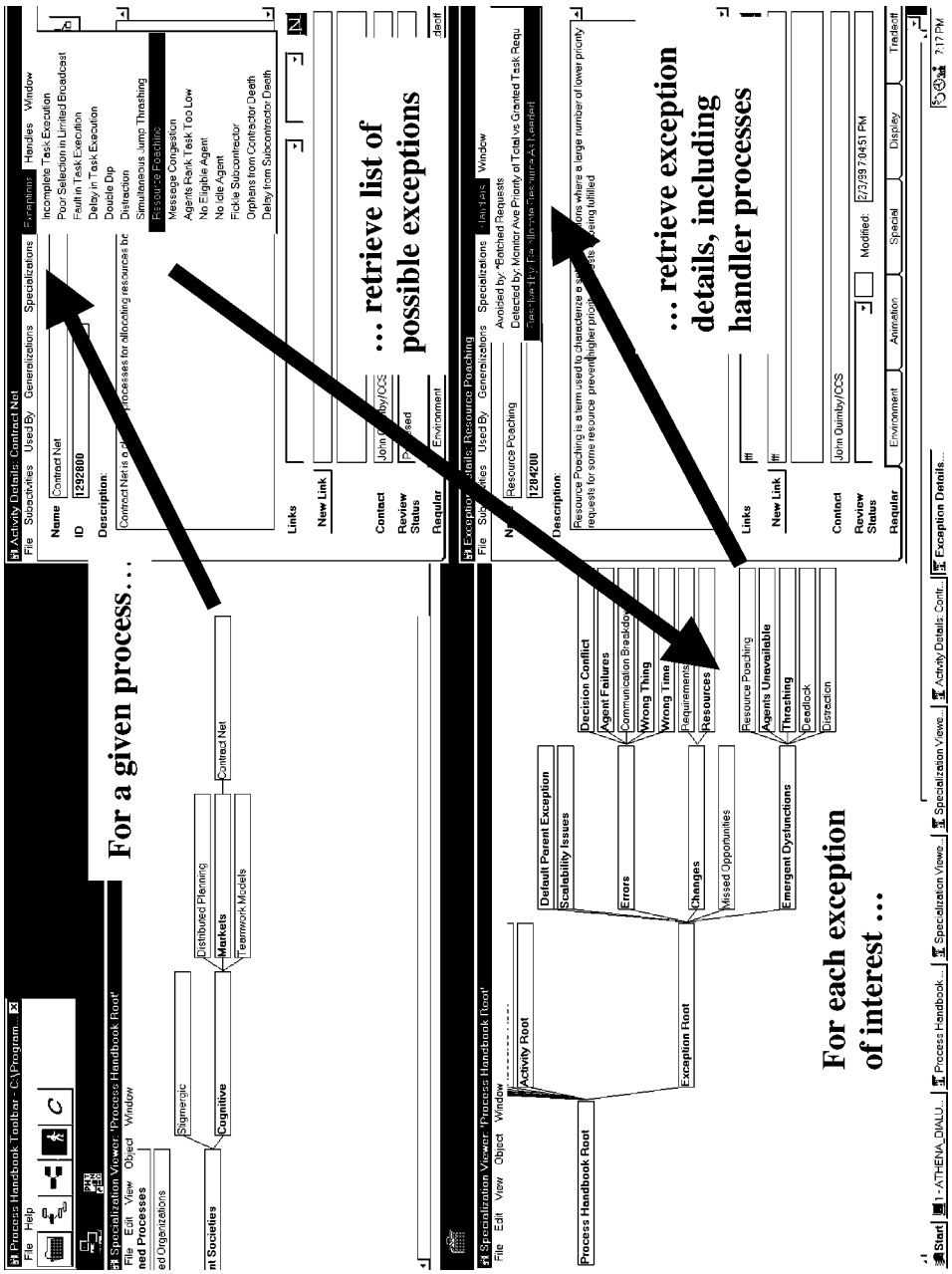

Figure 11. Screen shot of exception handling repository tool.

References

[1] P. Barthelmess and J. Wainer, Workflow systems: A few definitions and a few suggestions, in: *Proc. Conf. On Organizational Computing Systems (COOCS'95)* (1995) pp. 138–147.
[2] W.J. Clancey, Heuristic classification, Artificial Intelligence 27(3) (1985) 289–350.
[3] C. Dellarocas, J. Lee, T.W. Malone, K. Crowston and B. Pentland, Using a process handbook to design organizational processes, in: *Proceedings of the AAAI 1994 Spring Symposium on Computational Organization Design*, Stanford, CA (1994) pp. 50–56.
[4] C.A. Ellis, K. Keddara and G. Rozenberg, Dynamic change within workflow systems, in: *Proc. Conf. on Organizational Computing Systems (COOCS'95)* (1995) pp. 10–21.
[5] S. Fay, *The Collapse of Barings* (W.W. Norton, New York, 1997).
[6] M. Klein, Conflict resolution in cooperative design, University of Illinois at Urbana–Champaign, Technical Report UIUCDCS-R-89-1557.
[7] M. Klein, Supporting conflict resolution in cooperative design systems, IEEE Transactions on Systems, Man and Cybernetics 21(6) (1991) 1379–1390.
[8] M. Klein, An exception handling approach to enhancing consistency, completeness and correctness in collaborative requirements capture, Concurrent Engineering: Research and Applications 5(1) (1997) 37–46.
[9] T.W. Malone, K. Crowston, J. Lee and B. Pentland, Tools for inventing organizations: Toward a handbook of organizational processes, in: *Proceedings of 2nd IEEE Workshop on Enabling Tech. Infrastructure for Collaborative Enterprises* (1993) pp. 72–82.
[10] T.W. Malone, K. Crowston, J. Lee, B. Pentland, C. Dellarocas, G. Wyner, J. Quimby, C. Osborne and A. Bernstein, Toward a handbook of organizational processes, MIT Center for Coordination Science, Working Paper 198 (1997). To appear in *Management Science*.
[11] D. Raheja, Software system failure mode and effects analysis (SSFMEA) – A tool for reliability growth, in: *Proceedings of the Int'l Symp. on Reliability and Maintainability (ISRM'90)*, Tokyo, Japan (1990) pp. 271–277.
[12] P.G. Zhang, *Barings Bankruptcy and Financial Derivatives* (World Scientific Publishing Co, Singapore, 1995).