A Knowledge-Based Approach for Handling Exceptions in Business Processes

Chrysanthos Dellarocas Mark Klein

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

ABSTRACT

This paper describes a novel knowledge-based approach for helping business process designers and participants better manage exceptions (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 exploiting a generic and reusable body of knowledge describing 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.

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.

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 be built in order to limit the complexity of dealing with exceptions.

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 an *encoded* 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 process analysis tools, which assist designers to analyze "ideal" process models, 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 encoded knowledge base of generic strategies for detecting, diagnosing and resolving exceptions. The remainder of the paper will discuss how this approach works, how it relates to our previous work in this area, and 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 determine, 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 process elements 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 three principal elements of business process

models: activities, resources and constraints (describing goals and assumptions). Table 1 lists some examples.

Exceptions related to constraints	- 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

Table 1. A Subset of Exception Causes.

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 and constraint taxonomies can be defined in a similar manner.

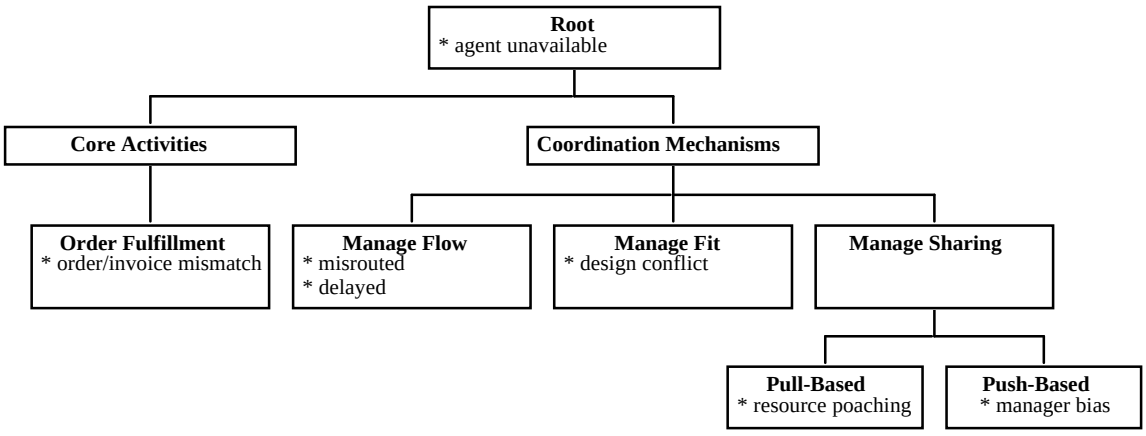

Figure 1. An Example of a Generic Activity Taxonomy with Failure Modes.

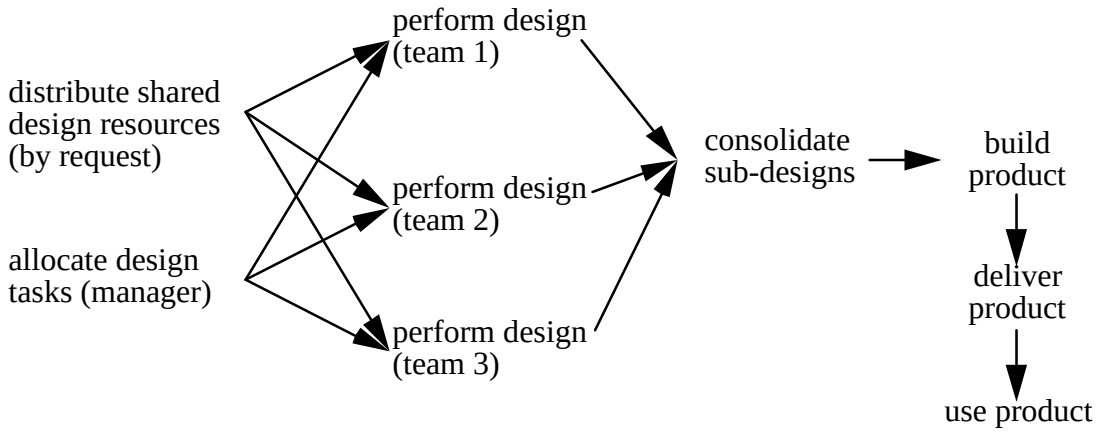

Figure 2. An Example “Ideal” Process Model.

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.

Given an “ideal” process model, to identify failure modes we need only identify the generic process element templates that match each element (activity, resource, constraint) 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.

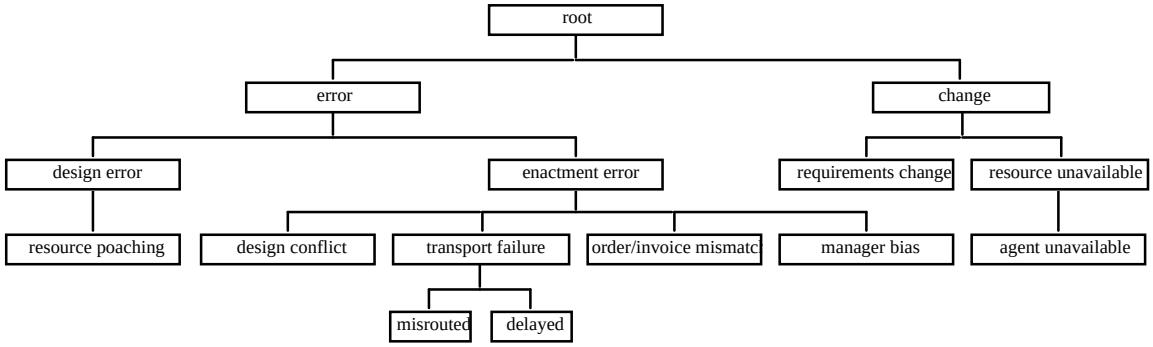

Figure 3. A Subset of the Exception Type Taxonomy.

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 an *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 response

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).

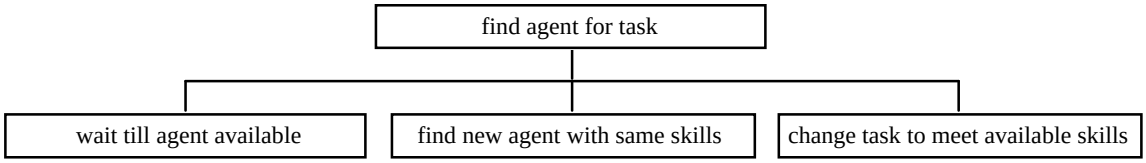

Figure 4. A Fragment of the Resolution Process Taxonomy.

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, constraints) and a taxonomy of exception types.

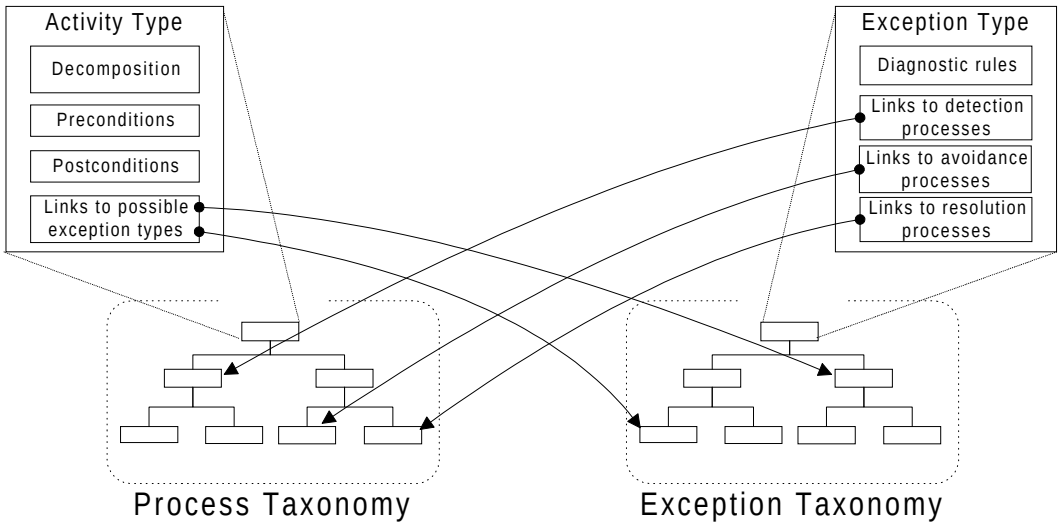

Figure 5. Overview of Exception Handling Knowledge Structures.

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.

3. 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 [5].

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.

4. 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.

This paper has emphasized the use of our exception handling knowledge base as a decision support tool for humans. Our ongoing work is primarily 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/

5. REFERENCES

1. P. Barthelmess and J. Wainer. Workflow Systems: a few Definitions and a few Suggestions. *Proc. Conf. on Organizational Computing Systems (COOCS'95)*, Aug. 13-16, 1995, pp. 138-147.
2. W. J. Clancey. Heuristic Classification. *Artificial Intelligence*, 27(3), 1985, pp. 289-350.
3. C. Dellarocas, J. Lee, T.W. Malone, K. Crowston and B. Pentland. Using a Process Handbook to Design Organizational Processes. *Proceedings of the AAAI 1994 Spring Symposium on Computational Organization Design*, Stanford, CA, March 21-23, 1994, pp. 50-56.
4. C.A. Ellis, K. Keddara and G. Rozenberg. Dynamic Change Within Workflow Systems. *Proc. Conf. On Organizational Computing Systems, (COOCS'95)*, Aug. 13-16, 1995, pp. 10-21.
5. G. Herman, M. Klein, et al. A Template-Based Process Redesign Methodology Based on the Process Handbook. MIT Center for Coordination Science Working Paper #tba
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), June 1991, pp. 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), March 1997, pp. 37-46.
9. T.W. Malone, K. Crowston, J. Lee and B. Pentland. Tools for Inventing Organizations: Toward a Handbook of Organizational Processes, *Proceedings of 2nd IEEE Workshop on Enabling Tech. Infrastructure for Collaborative Enterprises*, April 20-22, 1993, pp.72-82.
10. T.W. Malone, et al. Toward a handbook of organizational processes. MIT Center for Coordination Science Working Paper 198, January 1997. To appear in *Management Science*.
11. D. Raheja. Software system failure mode and effects analysis (SSFMEA)-a tool for reliability growth. *Proceedings of the Int'l Symp. on Reliability and Maintainability (ISRM'90)*, Tokyo, Japan, June 1990, pp. 271-277.